Markt&Technik

Frank Riemenschneider

PORTFOLIO Programmierpraxis



Von der Adressierung bis zur Business-Grafik in Turbo Pascal, Turbo Assembler und C



Die Begleitdiskette enthält alle Beispielprogramme

Portfolio
Programmierpraxis

PORTFOLIO Programmierpraxis

Frank Riemenschneider

Von der Adressierung bis zur Business-Grafik in Turbo Pascal, Turbo Assembler und C

Markt&Technik Verlag AG

CIP-Titelaufnahme der Deutschen Bibliothek

Riemenschneider, Frank:
Portfolio-Programmierpraxis : von der Adressierung bis zur Business-Grafik in Turbo Pascal,
Turbo Assembler und C / Frank Riemenschneider. –
Haar bei München : Markt-und-Technik-Verl., 1991
ISBN 3-87791-020-3

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

94 93 92 91

ISBN 3-87791-020-3

Einbandgestaltung: Grafikdesign Heinz Rauner
Dieses Produkt wurde mit Desktop-Publishing-Programmen erstellt
und auf der Linotronic 300 belichtet.
Lektorat: Baumann & Partner
Druck: Schoder, Gersthofen
Printed in Germany

Inhaltsverzeichnis

Vorwort

Als der Atari Portfolio im Oktober 1989 in Deutschland auf den Markt kam, konnte man von einer Sensation schlechthin sprechen: Zum erstenmal war es gelungen, einen Rechner im Taschenformat herzustellen, der nicht nur wie alle bis dahin erschienenen Konkurrenzprodukte einige Applikationen wie Taschenrechner und ein eingebautes Basic mitbrachte, sondern kompatibel zum IBM-Standard war. Man konnte also davon ausgehen, daß ein schier unerschöpfliches Potential von PC-Software zur Verfügung stehen würde, welches dieses Gerät in der Pocket-Klasse zum uneingeschränkten Marktführer werden lassen könnte. Auch die Erstellung eigener Software sollte dank komfortabler PC-Entwicklungssysteme ein Kinderspiel sein.

Soweit die Theorie. Als ich Ende 1989 im Auftrag der B&B Autoelektronik Vertriebs KG in Hannover mit der Entwicklung des elektronischen Fahrtenbuches Car-Office für PKW begann, wurde ich jedoch schnell eines besseren belehrt. Aufgrund seiner Abmessungen muß der Portfolio gewisse Hardware-Differenzen zum PC aufweisen, die sich leider auch auf die Software, insbesondere das Betriebssystem auswirken, so daß viele PC-Programme auf dem Portfolio nicht korrekt laufen, ja sogar Systemabstürze provozieren. Besonders der LCD-Bildschirm und der für heutige Verhältnisse kleine Speicher erfordern Portfolio-spezifische Programmierung. Aber auch das Fehlen einiger PC-Betriebssystem-Funktionen macht die Arbeit nicht gerade einfacher.

Der Hersteller und Entwickler des Gerätes, die englische Firma DIP (Atari baut den Rechner nur in Lizenz!), ist jedoch nicht bereit, wesentliche Informationen herauszurücken, wahrscheinlich, um den Nachbau des Gerätes unmöglich zu machen. Ich war deshalb gezwungen, in mühevoller Kleinarbeit die wesentlichen Details des Portfolio Byte für Byte selbst herauszufinden, da auch von dritter Seite bis heute keinerlei Literatur über die Programmierung des Portfolio existiert.

In dem vorliegenden Buch habe ich zusammen mit dem C-Spezialisten Michael Schuschk alle für mich wichtigen Gesichtspunkte der Portfolio-Programmierung in Turbo Pascal, Turbo C und Turbo Assembler herausgearbeitet, die es Ihnen einfach machen sollten, diesen Mini-PC zu programmieren. So werden z.B. nicht nur Routinen zur Bildschirmprogrammierung im Text- und Grafikmodus vorgestellt, sondern auch auf die RAM-Speicherkarten, das serielle Interface und sogar TSR-Programme wird eingegangen. Anhand zahlreicher Beispielprogramme (Chart-Grafik-Programm, Backup-Programm für Magnetkarten, Capture-Programm,

RAM-Kartenverwaltung etc.) werden die theoretischen Kenntnisse sofort in der Praxis demonstriert. Die vorgestellten Routinen, wie z.B. eine Pascal-Grafik-Unit, können leicht in eigene Programme eingebaut werden, so daß Sie sich über den Portfolio-spezifischen Teil Ihrer Programme keine Gedanken mehr zu machen brauchen. Da die Funktionen aufrufkompatibel zu entsprechenden PC-Routinen sind, können Ihre Programme auf dem PC ausgetestet und erst dann auf den Portfolio übertragen werden. Im Anhang finden Sie wichtige Hintergrundinformationen, wie z.B. alle Portfolio-Interrupts.

Obwohl ich bislang kein Anhänger von endlosen Danksagungen in Vorworten war, möchte ich in diesem Fall eine Ausnahme machen und der Firma COM-DATA dafür danken, daß sie als einziger Händler in Hannover die benötigte Portfolio-Peripherie genau dann vorrätig hatte, als ich sie brauchte, und einen von mir zerstörten Portfolio so unbürokratisch umtauschte.

Ich hoffe, daß Sie von meinen zahlreichen Nachtsitzungen mit dem Portfolio profitieren können (insgesamt stecken über 2000 Stunden Portfolio-Erfahrung in diesem Buch) und wünsche Ihnen viel Erfolg bei seiner Programmierung.

Frank Riemenschneider

Kapitel 1: Unterschiede zwischen PC und Portfolio

In den folgenden Abschnitten möchte ich auf die für die Programmierung des Portfolio wesentlichen Unterschiede zum PC eingehen.

Bitte haben Sie Verständnis dafür, daß ich hier keine Systembeschreibung eines PC geben kann, da das Buch sonst leicht 1000 Seiten oder mehr aufweisen würde. Im Anhang sind entsprechende Literaturhinweise aufgeführt.

1.1 Der Adreßraum

Der im PC und Portfolio verwendete Prozessor 8088 besitzt einen Adreßbus von 16 Bit Breite. Damit ist es möglich, einen Speicherbereich von 2^{16} = 65536 Byte zu adressieren. Galt dieser Wert zu Zeiten eines C64 noch als Sensation, so sahen die Entwickler des 8088, daß dieser Speicher für anspruchsvolle Anwendungen nie und nimmer ausreichend sein würde.

Da es zu dieser Zeit jedoch technisch noch schwierig bis unmöglich war, einen breiten Adreßbus zu bauen, bediente man sich eines Tricks, der sogenannten Segmentierung. Hierbei wird die physikalische Adresse durch zwei jeweils 16 Bit große Anteile gebildet, wobei der erste Anteil Segment genannt wird, der zweite Anteil Offset.

Diese beiden Anteile werden nun nicht einfach addiert, was höchstens einen 17-Bit-Wert ergeben könnte, sondern die 16 Bit des Segments werden um 4 Bit nach links verschoben, was einer Multiplikation mit 16 gleichkommt, erst dann wird der Offset aufaddiert. So ergibt sich insgesamt ein 20-Bit-Wert, womit 2^{20} = 1048576 Byte (1 Megabyte) adressiert werden können:

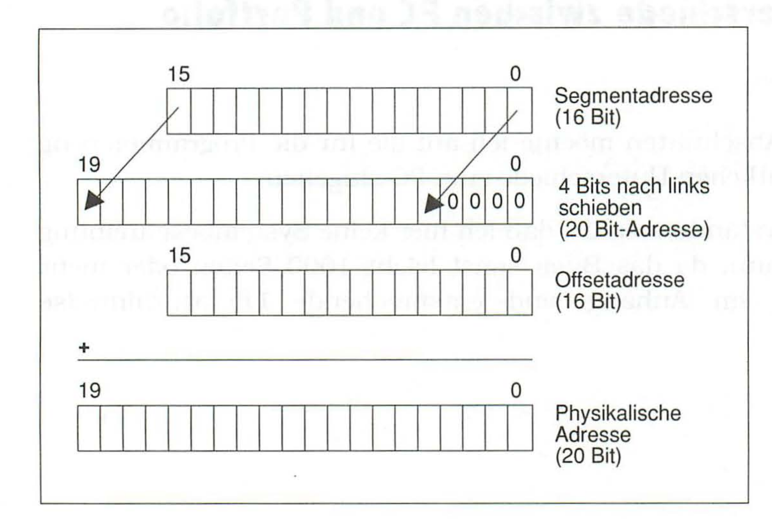


Bild 1.1: Bildung der Adresse über Segment und Offset

Diese Größenordnung bewegt sich schon in einem anderen Rahmen als die ursprünglichen 64 Kbyte, der Nachteil der Methode ist jedoch in der wesentlich verringerten Zugriffsgeschwindigkeit auf den Speicher zu sehen, da die physikalische Adresse erst aus Segment und Offset gebildet werden muß. Als Schreibweise für eine durch Segmentierung gewonnene Adresse hat sich durchgesetzt, erst den Segmentanteil und dann, durch einen Doppelpunkt getrennt, den Offsetanteil zu schreiben. So würde z.B. der Ausdruck

$E000:$0224

die Speicherzelle $E000*16+$0224 = $E0224 = 918052 adressieren. Die Schreibweise ist etwas verwirrend, da im Speicher selbst erst der Offset- und dann der Segmentanteil abgelegt wird. So würde der obige Ausdruck im Speicher als Bytefolge

$24 $02 $00 $E0

abgelegt, wobei noch zu beachten ist, daß bei 16-Bit-Wörtern immer erst die niederwertigen 8 Bit und dann die höherwertigen gespeichert werden. Dies ist ganz wichtig, wenn man auf irgendwelche Sprungvektoren (z.B. die der Interrupts) zugreift, da bei falscher Reihenfolge des Segment- und Offsetanteils und/oder bei Vertauschung des Low- und High-Byte der jeweiligen 16-Bit-Wörter natürlich an eine völlig falsche Adresse gesprungen wird, was einen Systemabsturz zur Folge haben kann.

Die Adreßbildung durch Segmentierung ist im Gegensatz zur linearen Adressierung nicht eindeutig. Da ein Segment an jeder durch 16 teilbaren

Speicherstelle beginnen kann, der Offset jedoch einen Bereich von 64 Kbyte abdeckt, kann in unserem Beispiel die Speicherstelle 918052 z.B. auch durch die Ausdrücke

$E022:$0004 oder $D023:$FFF4

beschrieben werden. Normalerweise müßte der Prozessor bei jedem Speicherzugriff (egal, ob es sich um Programmcode, Daten oder Stack handelt) Segment und Offset auslesen. Man kann sich nun leicht vorstellen, daß mit diesem Verfahren riesige und langsame Programme entständen, da der 8088 nur über einen 8-Bit-Datenbus verfügt. Um die 32 Bit von Segment und Offset zu lesen, wären also nicht weniger als vier (!) Speicherzugriffe erforderlich. Um dies zu vermeiden, wurden dem Prozessor vier Segmentregister spendiert, in denen sich die Segmentadressen von Code, Daten und Stack befinden. Alle Speicherzugriffe geschehen nun relativ zu diesen Segmentadressen, so daß grundsätzlich nur noch 16 Bit gelesen werden müssen. Ausschließlich bei Sprüngen ist die Möglichkeit vorhanden, auch das Segment der Zieladresse explizit anzugeben, ohne daß ein Segmentregister als Basis verwendet würde.

Nach diesem kleinen Exkurs in die Adreßbildung des 8088 (Experten mögen mir verzeihen ...) soll nun auf die tatsächliche Belegung der adressierbaren Bytes eingegangen werden.

Zunächst fällt auf, daß der Portfolio auch bei vollem Speicherausbau vier Kbyte weniger RAM besitzt als der PC. Dies liegt daran, daß das Video-RAM ab $B000:$0000 beim Portfolio von einem Teil eines System-RAM-Chips zur Verfügung gestellt wird, während der PC hierfür ein separates RAM aufweist, was schon deshalb erforderlich ist, weil sein Video-RAM 64 Kbyte umfaßt. Der Bereich ab $A000:$0000 liegt beim Portfolio brach, da hier natürlich keine zusätzliche Grafikkarte installiert werden kann. Der PC läßt die Installation einer MDA-Monochrom-Karte und gleichzeitig die einer Farbgrafikkarte zu. Dabei benutzt die MDA-Karte die unteren 32 Kbyte des Video-RAM, während die Farbgrafikkarte die oberen 32 Kbyte belegt. Natürlich werden tatsächlich nur 4 Kbyte (MDA) und 16 Kbyte (Grafik) benutzt, so daß beim PC mindestens 44 Kbyte des Video-RAM unbenutzt bleiben. Anders beim Portfolio: Er besitzt gerade die 4 Kbyte, die für die Textdarstellung im 80x25-Modus bzw. die Grafikdarstellung im 240x64-Modus erforderlich sind (mehr hierzu in Abschnitt 1.2).

F000 : FFFF
F000 : 0000 SYSTEM ROM 2
E000 : 0000
SYSTEM ROM 1 oder
D000 : 0000
SPEICHERKARTEN A: /B:
C000 : 0000
Kopie des VIDEO-RAMS
B100 : 0000
VIDEO-RAM (MDA-kompatibel)
B000 : 0000
Nicht belegt
A000 : 0000
Nicht belegt
9F00 : 0000
RAM-ERWEITERUNG
3F00 : 0000
SYSTEM-RAM (124 KBYTE)
0000 : 0000
Adreßraum PORTFOLIO

F000 : FFFF
BIOS-ROM
F000 : 0000
E000 : 0000 Speicherplatz für ROM-Erweiterungen
D000 : 0000
Speicherplatz für zusätzliches ROM-BIOS
C000 : 0000
VIDEO-RAM
B000 : 0000
Speicherplatz für zusätzliches VIDEO-RAM
A000 : 0000
RAM-ERWEITERUNG
80000 : 0000
SYSTEM-RAM (512 KBYTE)
0000 : 0000
Adreßraum PC

Bild 1.2: Vergleich der Adreßräume des PC und des Portfolio

Der PC besitzt serienmäßig nur 64 Kbyte ROM, welches das BIOS enthält. Das DOS, der Kommandoprozessor, die Gerätetreiber etc. liegen als Files vor, die beim Booten des Systems eingeladen werden und somit Speicherplatz im RAM benötigen. Dies ist beim Portfolio nicht der Fall: Nicht nur das BIOS, sondern auch das DOS, der Kommandoprozessor, die Gerätetreiber und alle Applikationen sind im ROM verankert und benötigen somit nur RAM-Speicher für die Systemvariablen, den Tastaturpuffer etc., also für die Dinge, die ständig modifiziert werden müssen. Da natürlich viel Platz für den Programmcode benötigt wird, besitzt der Portfolio insgesamt 256 Kbyte ROM, wobei in den unteren 128 Kbyte ab $C000:$0000 ausschließlich Applikationen enthalten sind, während in den oberen 128 Kbyte ab $E000:$0000 nur noch wenig Applikationssoftware zusammen mit dem BIOS, DOS, dem Kommandoprozessor und den Gerätetreibern verankert ist. Der Vorteil dieser Methode ist darin zu sehen, daß kaum System-RAM benötigt wird und das System in kürzester Zeit gebootet werden kann. Der eklatante Nachteil besteht in der Unveränderbarkeit des Codes: Alle Programmierfehler innerhalb des Systems (davon gibt es reichlich!) bleiben bis zu einer neuen Version erhalten. Auch ein Aufruf des

Kommandoprozessors ist von anderen Programmen aus nicht möglich, da ein File COMMAND.COM natürlich nicht existiert. Ein möglicher direkter Einsprung in das ROM verbietet sich von selbst, da man aus dem Kommandoprozessor nie wieder zurück in das aufrufende Programm käme. Wenigstens besteht die Möglichkeit, über sogenannte ROM-Extensions das BIOS und DOS zu erweitern bzw. zu modifizieren. Auch eigene Gerätetreiber können eingebunden werden, wie man es vom PC kennt (.SYS-File), wobei der entsprechende ROM-Treiber automatisch ersetzt wird.

Der Adreßbereich von $C000:0000 bis $C000:$FFFF ist beim Portfolio doppelt belegt: Durch Bank-Switching kann das Applikations-ROM ausgeblendet und hierfür eine RAM-Speicherkarte (im folgenden CCM für *Credit Card Memory* genannt) in den Adreßbereich des Prozessors gebracht werden. Für die Umschaltung des Prozessorports steht ein eigener Software-Interrupt zur Verfügung (siehe hierzu Abschnitt 1.4). Bei eingeblendetem CCM hat der Aufruf einer Applikation verheerende Folgen: Statt des Applikationscodes wird ein (nicht vorhandener) Programmcode auf dem CCM aufgerufen, was meistens den Absturz des Rechners zur Folge hat. Man sollte es sich daher angewöhnen, sofort nach Zugriff auf das CCM das ROM wieder einzublenden, damit einem solche Überraschungen erspart bleiben.

Man sieht, daß der Portfolio annähernd gleich aufgebaut ist wie der PC. Allerdings kann man daraus nicht schließen, daß alle PC-Programme auch auf dem Portfolio laufen, da die Hardwareadressen völlig unterschiedlich sind. Da immer mehr Programme aus Geschwindigkeitsgründen direkt auf die Hardware zugreifen (z.B. die CRT-Unit von Turbo Pascal), erlebt man die eine oder andere sehr unangenehme Überraschung. Dazu aber später in Kapitel 2.

Zum Schluß noch ein Wort zu möglichen Erweiterungen: Absolut unbenutzt ist in jedem Fall der Bereich von $9F00:$0000 bis $A000:$FFFF. Hier könnten z.B. über den Expansions-Port noch weitere RAM-Bausteine eingeblendet werden. Bei externen Geräten muß man jedoch aufpassen: Da der Portfolio im Gegensatz zum PC keinen programmierbaren Interrupt-Controller beinhaltet, existiert nur eine Interrupt-Leitung, die sich alle Geräte teilen müssen. Wenn Sie z.B. eine serielle Schnittstelle angeschlossen haben, welche die Leitung zum interruptgesteuerten Datenempfang benötigt, haben Sie keine Möglichkeit mehr, noch ein weiteres Gerät anzuschließen, das ebenfalls auf die Interruptleitung angewiesen ist. Diese Tatsache wertet meiner Meinung nach den Portfolio weit mehr ab als

der kleine Bildschirm oder der geringe Speicher. Die einzige Lösungsmöglichkeit des Problems besteht darin, einen eigenen Interruptcontroller zu installieren, dessen Ausgang dann an die Interrupt-Leitung des Portfolio geschaltet wird. Allerdings können Geräte dann nicht mehr direkt an den Systembus angeschlossen werden, so daß insgesamt das Fazit zu ziehen ist, daß dieser Engpaß nur durch einen enormen Hard- und Softwareaufwand zu beseitigen ist.

1.2 Die Interrupts des Portfolio

Bei kaum einer anderen Prozessorfamilie spielen die Interrupts eine derart dominierende Rolle wie bei den 80XX-Prozessoren, bieten Sie dem Programmierer doch Zugang zum Betriebssystem (DOS) und zum BIOS. Bevor ich auf die Unterschiede der einzelnen Interrupts eingehe, lassen Sie mich einige grundsätzliche Dinge zum Thema Interrupt sagen, da gerade dieser Bereich der Programmierung für viele Leute immer noch etwas Mystisches beinhaltet, das nur einigen wenigen Eingeweihten bekannt ist.

Dabei ist die Sache wirklich einfach: Dem Prozessor wird mitgeteilt, daß das laufende Maschinenprogramm unterbrochen werden soll. Dies kann durch einen Impuls an einer sogenannten Interrupt-Leitung des Prozessors geschehen (Hardware-Interrupt) oder durch eine Assembleranweisung, den Befehl INT (Software-Interrupt). In beiden Fällen reagiert der Prozessor gleich: Zunächst wird der gerade bearbeitete Maschinenbefehl bis zum Ende abgearbeitet, dann wird das Flag-Register auf den Stack gebracht. Warum dies? Nun, stellen Sie sich folgende Assemblersequenz vor:

```
...
sub ax,10
jz lOOP1
...
```

Nehmen wir an, während der Abarbeitung des SUB-Befehls würde ein Hardware-Interrupt ausgelöst, was z.B. durch die Tastatur möglich wäre. Falls das Flag-Register nicht gerettet würde, könnte man fast mit Sicherheit davon ausgehen, daß es nach der Rückkehr in das Hauptprogramm einen anderen Wert aufweist als vorher. Der bedingte Sprung, der nun folgt, würde also seine Entscheidung aufgrund eines veränderten Zero-Flags treffen, was zu einer falschen Programmfunktion führen würde. Da

fast jeder Maschinenbefehl das Flag-Register beeinflußt, ist es sicherlich sinnvoll, dies automatisch zu retten.

Jetzt wird der Programmzeiger auf den Stack gebracht, damit der Prozessor später weiß, wo er das unterbrochene Programm fortsetzen muß. Als nächstes wird dann in die Interrupt-Service-Routine gesprungen, die nichts anderes als ein normales Maschinenprogramm darstellt, das mit dem Befehl IRET abgeschlossen wird. Dieser Befehl bewirkt folgendes: Das Flag-Register und die Rücksprungadresse werden wieder vom Stack geholt und das unterbrochene Programm kann fortgesetzt werden; es hat von dem Interrupt-Aufruf praktisch nichts mitbekommen, wenn Sie als Interrupt-Programmierer nicht einen typischen Fehler gemacht haben: Im Gegensatz zu dem Flag-Register werden die übrigen Register nämlich nicht automatisch gerettet, was sinnvoll ist, da der Prozessor ja gar nicht wissen kann, welche Register in der Interrupt-Service-Routine benutzt werden. Man muß daher alle benutzten Register per Hand auf den Stack retten und sie vor dem IRET-Befehl wieder herunterholen.

Die Interrupt-Vektor-Tabelle

Bislang wurde gesagt, daß in eine Interrupt-Service-Routine gesprungen wird. Die berechtigte Frage ist, wo diese denn zu finden ist. Eine weitere Frage ist, wie man zwischen einzelnen Interrupt-Aufrufen differenzieren kann, denn bei einem Tastatur-Interrupt ist natürlich eine andere Reaktion angebracht als bei einem Disketten-Fehler. Um hier unterscheiden zu können, hat man einfach 256 verschiedene Interrupts definiert, was für alle Zwecke ausreichen dürfte. Bei den Software-Interrupts wird einfach hinter dem INT-Befehl die entsprechende Interruptnummer angegeben, so daß der Prozessor anhand dieser Nummer den gewünschten Interrupt aufrufen kann. Schwieriger ist die Sache bei den Hardware-Interrupts: Hier ist für jeden Interrupt eine eigene Leitung des Prozessors erforderlich, weshalb sich die Anzahl der Hardware-Interrupts auch sehr in Grenzen hält (acht externe Interrupts, fünf Interrupts direkt vom Prozessor).

Die Adressen der einzelnen Interrupt-Service-Routinen holt sich der Prozessor aus der sogenannten Interrupt-Vektor-Tabelle. Sie beginnt an der untersten Stelle des Adreßraums, also bei $0000:$0000, und weist für jeden Interrupt einen 4 Byte langen Eintrag auf: die Segment- und Offsetadresse der Interrupt-Service-Routine. Damit ist die Tabelle 4x256 = 1024 Byte lang, wobei dem Interrupt 0 die ersten 4 Byte ($0000:$0000 bis

$0000:$0003) zugeordnet sind, dem Interrupt 255 die letzten 4 Byte ($0000:$03FC bis $0000:$03FF). Um die korrespondierende Adresse eines bestimmten Interrupts zu finden, muß man nur seine Nummer mit 4 multiplizieren und erhält den Offset innerhalb der Interrupt-Vektor-Tabelle (z.B. $0008 für den Interrupt 2). Innerhalb eines Eintrags findet man zunächst den Offset (Low- dann High-Byte) und anschließend das Segment (Low- und Highbyte), wie es auch schon in Abschnitt 1.1 beschrieben wurde. Im Prinzip können übrigens alle 256 Interrupts vom Programm aus mit einem INT-Befehl aufgerufen werden, also auch die Hardware-Interrupts. So wird z.B. über den Interrupt 5 eine Hardcopy des aktuellen Bildschirms gedruckt. Normalerweise ist dieser Interrupt als Hardware-Interrupt konzipiert, der auf die Tasten [Fn] und [P] reagiert. Niemand kann Sie jedoch daran hindern, diesen Interrupt von einem Programm aus mit dem Befehl INT 5 aufzurufen, wenn Sie eine Hardcopy wünschen. Bei den meisten Hardware-Interrupts ist ein softwaremäßiger Aufruf jedoch nicht sehr ratsam.

Der Hintergedanke dieser Methode besteht darin, daß man eine Tabelle im RAM natürlich leicht modifizieren und somit den Interruptvektor auf eine eigene Service-Routine umlenken kann. In weiser Vorraussicht hat z.B. DOS einen Hardware-Interrupt eingerichtet, der bei jedem Ablauf eines Timers ausgelöst wird, im PC 18,2mal in einer Sekunde (Interrupt-Nummer 8). Da dem DOS (bzw. seinen Erfindern) jedoch keine Verwendung eingefallen ist, schließt es die Service-Routine einfach mit einem IRET-Befehl ab. Einigen Programmieren ist jedoch eine Anwendung eingefallen, z.B. die ständige Ausgabe der Uhrzeit auf dem Bildschirm. Sie beschreiben die Interrupt-Vektor-Tabelle also ab der Stelle $0020 (4x8=32=$20) mit der Segment- und Offsetadresse einer eigenen Routine (man nennt dieses Verfahren »Verbiegen des Interruptvektors«) und können nun sicher sein, daß diese Routine im PC 18,2mal in der Sekunde aufgerufen wird. (Im Portfolio liegen die Dinge etwas anders, dazu aber später.) Ohne den Timer-Interrupt wäre eine solche Programmierung schwierig bis unmöglich, so daß man dankbar sein kann, daß die Möglichkeit besteht, auch wenn sie von DOS nicht genutzt wird.

Im folgenden habe ich alle Interrupts des Portfolio zusammengestellt, wobei ich in der Spalte »Adresse« nicht die Adresse des Interrupt-Vektors (die kann man ja einfach berechnen, s.o.), sondern die Adresse der tatsächlichen Interrupt-Service-Routine angegeben habe, also den Inhalt des Interrupt-Vektors. In der vorletzten Spalte ist die Art des Interrupt angegeben, wobei ein »S« für Software-Interrupt und ein »H« für Hardware-Interrupt steht. Ein »T« hingegen markiert einen Tabelleninterrupt. Dies ist

ebenfalls ein Vektor, der jedoch auf eine für DOS wichtige Tabelle zeigt (z.B. Interrupt $1E), also auf gar keinen Fall mit dem INT-Befehl aufgerufen werden darf. Mit einem »F« sind diejenigen Interrupts markiert, deren Vektor zwar auf eine Service-Routine zeigt, die jedoch nicht mit einem IRET-, sondern einem RET FAR-Befehl abgeschlossen sind. Dadurch wird zwar die Rücksprungadresse vom Stack geholt, nicht jedoch das Flag-Register. Dies muß »per Hand« nachgeholt werden, damit langfristig ein Stacküberlauf verhindert wird und die gleiche Situation wie vor dem Interrupt-Aufruf gegeben ist. Allein durch diese Konzeption bedingt sollte man aber einsehen, daß es nicht sinnvoll sein kann, diese Interrupts mit einem INT-Befehl aufzurufen.

In der letzten Spalte finden Sie Hinweise zu besonderen Interrupts: Zum einen besagt der Eintrag IRET, daß die Interrupt-Service-Routine nur aus einem einzigen Befehl besteht, nämlich dem Rücksprung-Befehl ins unterbrochene Programm. Der aufgerufende Interrupt bewirkt also überhaupt nichts! Einige andere Interrupts sind mit der Bemerkung INT gekennzeichnet. Hier handelt es sich um interne DOS- oder BIOS-Interrupts, die man möglichst nicht aufrufen sollte, da sie nicht dokumentiert sind und in späteren Betriebssystemversionen durchaus ihre Funktion ändern oder ganz verschwinden können. Bevor ich auf die Unterschiede zum PC eingehe, hier erst einmal die Tabelle mit allen Portfolio-Interrupts der BIOS-Version 1.052:

Interrupttabelle Atari-Portfolio BIOS 1.052

Nr.	Adresse	Funktion	IA	Bem.
$00	03D2:00CE	Division durch Null	H	
$01	E000:246D	Einzelschritt	H	IRET
$02	E000:1CB2	NMI (Fehler in Speicher)	H	
$03	E000:246D	Breakpoint erreicht	H	IRET
$04	E000:246D	Numerischer Überlauf	H	IRET
$05	E000:1F12	Hardcopy	H	
$06	0000:0000	unbelegt		
$07	0000:0000	unbelegt		
$08	E000:29F1	Timer	H	
$09	E000:14F1	Tastatur	H	
$0A	0000:0000	unbelegt		
...	....	....	..	....
$0F	0000:0000	unbelegt		

Interrupttabelle Atari-Portfolio BIOS 1.052

Nr.	Adresse	Funktion	IA	Bem.
$10	E000:2B8B	BIOS: Video-Funktionen	S	
$11	E000:1D60	BIOS: Konfiguration	S	
$12	E000:1CA6	BIOS: Speichergröße	S	
$13	E000:0AE5	BIOS: Magnetkarten/RAM-Disk	S	
$14	E000:2470	BIOS: Serielles Interface	S	
$15	E000:1C78	Keine Funktion	S	
$16	EB32:0069	BIOS: Tastaturabfrage	S	
$17	E000:1DAE	BIOS: Paralleles Interface	S	
$18	E000:0483	Keine Funktion	S	
$19	E000:0524	BIOS: Reset (Alt/Ctrl/Del)	S	
$1A	E000:0618	BIOS: Zeit/Datum ermitteln	S	
$1B	FF00:0562	Programmabbruch	S	
$1C	E000:246D	Aufruf nach jedem INT08	S	IRET
$1D	0000:0000	unbelegt		
$1E	0040:01F2	Disk-Parameter-Tabelle	T	
$1F	0000:0000	unbelegt		
$20	E527:3281	DOS: Programm beenden	S	
$21	E527:44DA	DOS: DOS-Funktionen	S	
$22	F24F:026C	Adresse DOS-Programmende	A	
$23	03D2:00D5	Adresse DOS Ctrl/Break	A	
$24	03D2:009D	Adresse DOS Fehler	A	
$25	E527:05B8	DOS: Magnetkarte lesen	S	
$26	E527:05BE	DOS: Magnetkarte schreiben	S	
$27	E527:32CC	DOS: Prg. resident beenden	S	
$28	E527:04B3	Aufruf bei Tastaturpolling	S	IRET
$29	FF00:0555	DOS: Zeichen ausgeben	S	INT
$2A	E527:04B3	Keine Funktion	S	IRET
$2B	0000:0000	unbelegt		
....	....	....	..	....
$3E	0000:0000	unbelegt		
$3F	03D2:00C6	Keine Funktion	S	
$40	0000:0000	unbelegt		
$41	0040:01F2	Disk-Parameter-Tabelle	T	
$42	0000:0000	unbelegt		
....	....	....	..	....
$49	0000:0000	unbelegt		

Interrupttabelle Atari-Portfolio BIOS 1.052				
Nr.	**Adresse**	**Funktion**	**IA**	**Bem.**
$4A	FF9A:0227	Alarmzeit erreicht	S	
$4B	0000:0000	unbelegt		
....	....	....	..	....
$5F	0000:0000	unbelegt		
$60	EA00:07E9	DIP: BIOS-Funktionen	S	INT
$61	E000:327F	DIP: BIOS-Funktionen	S	
$62	0000:0000	unbelegt		
....	....	....	..	....
$FF	0000:0000	unbelegt		

Bei der neuesten ausgelieferten BIOS-Version 1.072 haben sich praktisch alle Interrupt-Adressen geändert, so daß hierfür folgende Tabelle gültig ist:

Interrupttabelle Atari-Portfolio BIOS 1.072				
Nr.	**Adresse**	**Funktion**	**IA**	**Bem.**
$00	0380:00CE	Division durch Null	H	
$01	E000:24D7	Einzelschritt	H	IRET
$02	E000:1D06	NMI (Fehler in Speicher)	H	
$03	E000:24D7	Breakpoint erreicht	H	IRET
$04	E000:24D7	Numerischer Überlauf	H	IRET
$05	E000:1F6C	Hardcopy	H	
$06	0000:0000	unbelegt		
$07	0000:0000	unbelegt		
$08	E000:2A5B	Timer	H	
$09	E000:1545	Tastatur	H	
$0A	0000:0000	unbelegt		
...	....	....	..	....
$0F	0000:0000	unbelegt		
$10	E000:2BF5	BIOS: Video-Funktionen	S	
$11	E000:1DB4	BIOS: Konfiguration	S	
$12	E000:1CFA	BIOS: Speichergröße	S	
$13	E000:0B20	BIOS: Magnetkarten/RAM-Disk	S	
$14	E000:24DA	BIOS: Serielles Interface	S	
$15	E000:1CCC	Keine Funktion	S	

Interrupttabelle Atari-Portfolio BIOS 1.072				
Nr.	**Adresse**	**Funktion**	**IA**	**Bem.**
$16	EB3E:005E	BIOS: Tastaturabfrage	S	
$17	E000:1E02	BIOS: Paralleles Interface	S	
$18	E000:048F	Keine Funktion	S	
$19	E000:0530	BIOS: Reset (Alt/Ctrl/Del)	S	
$1A	E000:0653	BIOS: Zeit/Datum ermitteln	S	
$1B	FF00:0562	Programmabbruch	S	
$1C	E000:24D7	Aufruf nach jedem INT08	S	IRET
$1D	0000:0000	unbelegt		
$1E	0040:01F4	Disk-Parameter-Tabelle	T	
$1F	0000:0000	unbelegt		
$20	E52D:32E0	DOS: Programm beenden	S	
$21	E52D:451C	DOS: DOS-Funktionen	S	
$22	F25C:026D	Adresse DOS-Programmende	A	
$23	0380:00D5	Adresse DOS Ctrl/Break	A	
$24	0380:009D	Adresse DOS Fehler	A	
$25	E52D:05BF	DOS: Magnetkarte lesen	S	
$26	E52D:05C5	DOS: Magnetkarte schreiben	S	
$27	E52D:32FB	DOS: Prg. resident beenden	S	
$28	E52D:04BA	Aufruf bei Tastaturpolling	S	IRET
$29	FF00:0555	DOS: Zeichen ausgeben	S	INT
$2A	E52D:04BA	Keine Funktion	S	IRET
$2B	0000:0000	unbelegt		
....	....	....	..	....
$3E	0000:0000	unbelegt		
$3F	0380:00C6	Keine Funktion	S	
$40	0000:0000	unbelegt		
$41	0040:01F4	Disk-Parameter-Tabelle	T	
$42	0000:0000	unbelegt		
....	....	....	..	....
$49	0000:0000	unbelegt		
$4A	FF9F:0227	Alarmzeit erreicht	S	
$4B	0000:0000	unbelegt		
....	....	....	..	....
$5F	0000:0000	unbelegt		
$60	EA0C:07DE	DIP: BIOS-Funktionen	S	INT
$61	E000:32E6	DIP: BIOS-Funktionen	S	
$62	0000:0000	unbelegt		
....	....	....	..	....
$FF	0000:0000	unbelegt		

Die Belegung der Interrupts stimmt mit dem PC überein, wenn man von einigen Ausnahmen absieht: So sind beim Portfolio viele Interrupts unbelegt, weil einfach die entsprechende Hardware nicht vorhanden ist (z.B. Festplatten-Interrupt Nr. $0D). Auch der Interrupt der ersten seriellen Schnittstelle ist unbelegt, weil das Portfolio-Interface keinen festen Interrupt benutzt wie der PC, sondern im Prinzip jeden Interrupt belegen kann. (Dazu aber mehr in Abschnitt 1.6.) Beim PC enthält der Tabelleninterrupt $41 die Adresse der ersten Festplatten-Tabelle. Um diesen Interrupt nicht unbelegt zu lassen, hat man ihn beim Portfolio mit einer Kopie des Interrupt $1E (Disk-Parameter-Tabelle) belegt, wobei mit »Disk« die interne RAM-Disk sowie die CCMs gemeint sind. Der Interrupt $18 (ROM-Basic) ist zwar belegt, die Interrupt-Service-Routine gibt jedoch nur eine Meldung aus, daß kein ROM-Basic vorhanden ist.

Ein Unterschied mit weitreichenden Folgen besteht beim internen Interrupt $28. Man muß zunächst wissen, daß der Befehlsinterpreter (COMMAND.COM) im PC beim Warten auf eine Eingabe des Benutzers die Tastatur durch Polling abfragt, d.h., in einer Schleife wird ständig nachgesehen, ob eine Taste gedrückt wurde. Der Vorteil dieser Methode besteht darin, daß zwischendurch noch andere Aufgaben wahrgenommen werden können. Hierzu dient beim PC der Interrupt $28. Er sendet z.B. Zeichen an den Drucker, wenn durch den PRINT-Befehl eine Hintergrundausgabe gestartet wurde. Wichtig ist aber vor allen Dingen, daß die meisten TSR-Programme mit seiner Hilfe feststellen, ob das Hintergrundprogramm durch den Druck eines Hot-Keys unterbrochen werden darf. Auf dem Portfolio ist dieser Interrupt unbelegt, da der Befehlsinterpreter die Tastatur nicht durch Polling abfragt, sondern eine BIOS-Funktion benutzt, die wartet, bis eine Taste gedrückt wurde. Ist dies längere Zeit nicht der Fall, wird der Portfolio stufenweise heruntergeschaltet, was sich z.B. durch Abschalten des Bildschirms äußert. Mit diesem Verfahren ist es möglich, gegenüber dem Polling erhebliche Einsparungen an Strom zu erzielen, was im Gegensatz zum Portfolio beim PC ja keine Rolle spielt. Die unangenehme Konsequenz besteht nun darin, daß praktisch kein für den PC entwickeltes TSR-Programm auf dem Portfolio korrekt läuft, da der Interrupt $28 vom COMMAND.COM nicht aufgerufen wird. Durch trickreiche Programmierung ist es zwar möglich, auch für den Portfolio TSR-Programme zu entwickeln (siehe Kapitel 2.8), diese können aber wie gesagt nicht einfach vom PC übernommen werden.

Um die spezielle Hardware des Portfolio steuern zu können, hat die Firma DIP erweiterte BIOS-Funktionen zur Verfügung gestellt, die über den Interrupt $61 aufgerufen werden. Dieser Interrupt ist beim PC unbelegt

und führt zu Problemen bei der Entwicklung von Portfolio-Software auf dem PC, da dieser natürlich beim Aufruf des unbelegten Interrupt $61 abstürzt. DIP bietet daher ein selbstentwickeltes Emulationsprogramm für den PC an, das den Interrupt $61 belegt und die meisten Funktionen auch auf dem PC emulieren kann. Der Interrupt $60 ist beim PC ebenfalls unbelegt, dient aber auch beim Portfolio nur internen Zwecken, so daß man auf seinen Aufruf verzichten sollte.

1.3 Der LCD-Bildschirm

Der Portfolio stellt nur zwei Video-Modi zur Verfügung: Einen Textmodus in der Auflösung von 80x25 Zeichen, der erfreulicherweise MDA-kompatibel ist, sowie einen völlig inkompatiblen Grafikmodus in der Auflösung von 240x64 Grafikpunkten. Die beiden Video-Modi können über den BIOS-Interrupt $10 gesetzt werden, wobei der Modus $07 (Text) bzw. $0A (Grafik) verwendet werden muß. Alle anderen für den PC zulässigen Video-Modi (0 bis 6) sind beim Portfolio nicht anwendbar.

Zunächst zum Textmodus: Um nicht die gesamte PC-Software, die auf einem 80x25-Bildschirm basiert, für den Portfolio unbenutzbar zu machen, hat man sich bei DIP ein Verfahren überlegt, wie man trotz des Miniaturbildschirms, der nur 40x8 Zeichen darstellt, auch diese Programme ablaufen lassen kann. Zunächst einmal legte man das Video-RAM nach $B000:$0000, um einen MDA-kompatiblen Adreßraum zu haben, so daß auch Programme, die direkt in den Bildschirmspeicher schreiben, benutzt werden können. Für jedes Zeichen werden 2 Byte reserviert, so daß insgesamt 80x25x2 = 4000 Byte belegt werden. Das erste Byte entspricht dem ASCII-Code des Zeichens, das zweite dem sogenannten Attribut-Byte, welches beim PC die Eigenschaften der Darstellungsweise des Zeichens (z.B. invertiert, unterstrichen etc.) festlegt. Beim Portfolio hat das Attribut-Byte keinerlei Wirkung, da der LCD-Controller nicht in der Lage ist, andere als »normale« Zeichen darzustellen. Die unangenehme Folge davon ist, daß alle Programme, die z.B. mit Auswahlbalken etc. arbeiten und dazu natürlich invertierte Darstellungen benötigen (z.B. Turbo Pascal), nicht mehr korrekt funktionieren können, da der Auswahlbalken schlicht und einfach nicht sichtbar ist. Man muß diese Programme so umschreiben, daß sie z.B. einen Auswahlpfeil benutzen, auf keinen Fall aber irgendwelche Informationen mit Hilfe des Attribut-Bytes vermitteln.

Die Zeichen werden zeilenweise von oben nach unten im Video-RAM abgelegt, so daß der Zeilenanfang der Zeile 0 bei $B000:$0000 beginnt, der Zeilenanfang der Zeile 1 bei $B000:$00A0 usw. Innerhalb einer Zeile werden die Zeichen von links nach rechts gespeichert. Da der Bildschirm nur eine Breite von 240 Pixels aufweist, besitzt jedes Zeichen im Gegensatz zum PC-Zeichensatz nur eine Breite von 6 Pixels (gegenüber 8 beim PC), wodurch die Darstellungsqualität natürlich deutlich vermindert wird.

	Spalte								
Zeile	00	01	02	03	..	76	77	78	79
0	$000	$002	$004	$006	..	$098	$09A	$09C	$09E
1	$0A0	$0A2	$0A4	$0A6	..	$138	$13A	$13C	$13E
2	$140	$142	$144	$146	..	$1D8	$1DA	$1DC	$1DE
3	$1E0	$1E2	$1E4	$1E6	..	$278	$27A	$27C	$27E
4	$280	$282	$284	$286	..	$318	$31A	$31C	$31E
..	....	....	....	....	..	....	....	....	....
20	$C80	$C82	$C84	$C86	..	$D18	$D1A	$D1C	$D1E
21	$D20	$D22	$D24	$D26	..	$DB8	$DBA	$DBC	$DBE
22	$DC0	$DC2	$DC4	$DC6	..	$E58	$E5A	$E5C	$E5E
23	$E60	$E62	$E64	$E66	..	$EF8	$EFA	$EFC	$EFE
24	$F00	$F02	$F04	$F06	..	$F98	$F9A	$F9C	$F9E

Bild 1.3: Aufbau des MDA-kompatiblen Video-RAM

Da der Portfolio-Bildschirm jedoch nur eine Auflösung von 40x8 Zeichen aufweist, wird dieser einfach als Fenster innerhalb des 80x25-Bildschirms angesehen. Durch Scrollen mit den Cursortasten in Verbindung mit der [Alt]-Taste kann dieses Fenster dann innerhalb des großen Bildschirms bewegt werden. Die jeweilige Position des Fensters (des realen Portfolio-Bildschirms) wird durch die sogenannte virtuelle Bildschirmposition beschrieben, was nichts anderes heißt, als daß man zwei Koordinaten x-virtuell und y-virtuell definiert hat, welche die Position der linken, oberen Ecke des Fensters innerhalb des 80x25-Bildschirms festlegen:

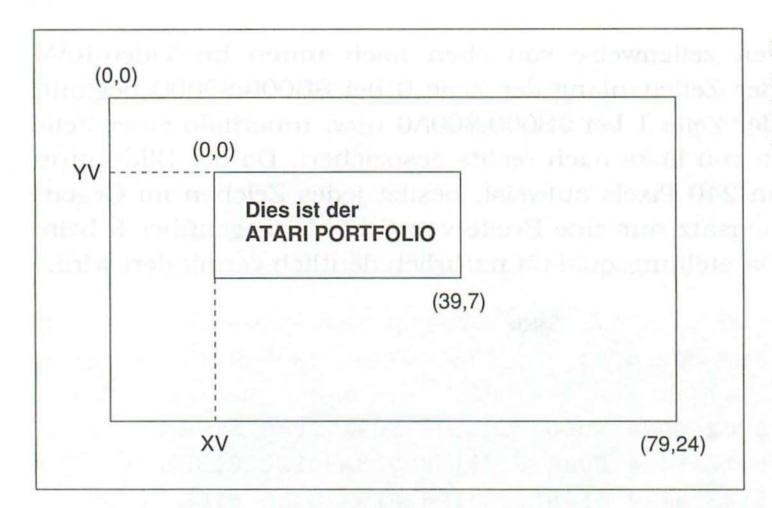


Bild 1.4: Definition des Portfolio-Bildschirms als Fenster

Der Portfolio besitzt zwei Funktionen des erweiterten BIOS-Interrupt $61, mit denen man die Fensterposition beeinflussen kann: Die Funktion $10 setzt es an eine absolute Position, wobei im AL-Register der Wert 1, im DH-Register die Zeile und im DL-Register die Spalte übergeben werden müssen. Man muß beachten, daß nur Zeilenwerte bis 17 (17+8=25) und Spaltenwerte bis 40 (40+40=80) zulässig sind. Mit derselben Funktion kann man die Position auch ermitteln, indem man im AL-Register den Wert 0 übergibt. Die Zeilen- und Spaltennummern werden in den Registern DH und DL zurückgeliefert.

Beispiel:
AH = $10, AL = $01, DH = $05, DL = $20, Aufruf Int $61
Das Fenster wird so gesetzt, daß seine linke obere Ecke an den Koordinaten (x=$20,y=$01) des 80*25-Bildschirms zu finden ist.

Durch die Interrupt-Funktion $11 kann man das Fenster relativ zu seiner aktuellen Position bewegen. Hierfür wird im DL-Register die Bewegungsrichtung festgelegt:

1: Bewegung nach oben
2: Bewegung nach unten
3: Bewegung nach links
4: Bewegung nach rechts

Im AL-Register wird die Anzahl der Zeilen (wenn DL = 0 oder DL =1) bzw. Spalten (wenn DL = 2 oder DL =3) übergeben, um die das Fenster bewegt werden soll.

Beispiel:
AH = $11, AL = $05, DL = $04, Aufruf INT $61
Das Fenster wird um 5 Spalten nach rechts bewegt.

Innerhalb des Textmodus kann zwischen drei Betriebsarten gewählt werden: Zum einen ist da der statische 80*25-Modus. Hier bleibt die Fensterposition immer an einer Position, wenn sie nicht durch die obigen beiden Interrupt-Funktionen (oder durch die Tastatur mit [Alt] plus Cursortaste) verändert wird. Die Bildschirmausgaben werden jedoch auf die Größe des Gesamtbildschirms (80*25) bezogen, so daß alle Ausgaben außerhalb des Fensters für den Benutzer unsichtbar bleiben. Er muß selbst dafür sorgen, daß das Fenster an die Position verschoben wird, an der sich für ihn interessante Bildschirmausgaben befinden.

Dagegen wird bei dem dynamischen Modus das Fenster automatisch mit den Bildschirmausgaben mitbewegt (genauer gesagt: Es folgt dem Cursor), so daß man anschließend auf jeden Fall immer den letzten Teil der Ausgabe im Fenster sichtbar hat. Der Sinn dieses Modus besteht darin, daß der Anwender die Texte während der Ausgabe mitlesen kann. Leider geschieht dies aber meistens so schnell, daß man in der Praxis keine Möglichkeit hat, mitzulesen und anschließend wie im statischen Modus das Fenster »per Hand« bewegen muß, um den gesamten Ausgabetext lesen zu können.

Der 40x8-Modus schließlich ist ein echter Portfolio-Modus und speziell nur für den Portfolio entwickelte Programme brauchbar. Hierbei wird der physikalische Bildschirm auf die tatsächlich sichtbaren 40x8 Zeichen reduziert, so daß eine Bildschirmausgabe nach jeweils 40 Zeichen eine neue Zeile beginnt. Da der 80x25-Bildschirm nicht mehr existiert, kann das Fenster, welches in diesem Modus eigentlich kein Fenster mehr ist, natürlich auch nicht mehr bewegt werden, weder durch die Interrupts noch durch die Cursortasten. Nach jweils 8 Zeilen wird der Bildschirm um eine Zeile nach oben gerollt, was in den anderen zwei Modi erst nach 25 Zeilen der Fall ist. Die zulässigen Spalten- und Zeilenangaben z.B. für die Positionierung des Cursors sind ebenfalls auf die Werte 0–7 (Zeile) und 0–39 (Spalte) reduziert. Dieser Modus sollte von allen spezifischen Portfolio-Programmen benutzt werden, um das lästige Rollen des Fensters zu vermeiden.

Auch für das Setzen/Auslesen der Betriebsarten stellt der Interrupt $61 eine Funktion ($0E) bereit. Um die neue Betriebsart zu setzen, muß im AL-Register der Wert 1 und im DL-Register das zugehörige Byte nach folgender Tabelle übergeben werden:

DL = 0: Statischer Modus (80x25 Zeichen, realer Bildschirm stellt 40x8-Fenster dar)
DL = 1: Portfolio-Modus (40x8 Zeichen)
DL = 2: Dynamischer Modus (80x25, realer Bildschirm stellt 40x8-Fenster dar)
(DL = 128: Grafikmodus, hier nicht interessant)

Der Interrupt gibt im DL-Register den alten Betriebsmodus zurück. Falls im AL-Register der Wert 0 übergeben wird, wird kein neuer Modus gesetzt, sondern der aktuelle im DL-Register zurückgeliefert.

Beispiel:
AH = $0E, AL = $01, DL = $02, Aufruf INT $61
Der dynamische Bildschirm-Modus wird aktiviert.

Ich möchte Sie nun mit einem weiteren, nicht unerheblichen Problem bei der Bildschirmprogrammierung des Portfolio konfrontieren, dem Screen-Refresh. Zunächst muß man sich klarmachen, daß das Video-RAM nichts anderes ist als ein Stück gewöhnlicher Speicher, aus dem der Video-Controller seine Informationen ausliest. Wenn man nun irgendwelche Informationen in das Video-RAM schreibt, ist es noch längst nicht selbstverständlich, daß diese sofort auf dem Bildschirm erscheinen. Hierzu ist ein Refresh erforderlich, d.h., der Video-Controller aktualisiert den Bildschirminhalt nach den neuesten Informationen, die er aus dem Video-RAM erhält. Beim PC muß man sich keine Sorgen um den Refresh machen, da dieser automatisch regelmäßig durchgeführt wird. Nicht so beim Portfolio: Wenn Sie hier direkt in das Video-RAM hineinschreiben, müssen Sie dem LCD-Controller explizit mitteilen, daß er doch bitte den Bildschirm »refreshen« möchte, damit Ihre Informationen auch auf ihm erscheinen. Normalerweise geht dies nur durch direkte Hardware-Programmierung des LCD-Controllers, aber auch hier hat DIP eine Interrupt-Funktion ($12) implementiert, die den Refresh durchführt. Dabei müssen keine Parameter in den Registern übergeben werden:

AH = $12, Aufruf INT $61: Screen Refresh

In eigenen Programmen kann man auf diese Tatsache Rücksicht nehmen, indem man einfach nach jedem direkten Zugriff auf das Video-RAM diese Interrupt-Funktion aufruft. Was aber ist mit PC-Programmen? Nun, diese besitzen natürlich keinerlei Befehle zum Bildschirm-Refresh, da dies ja automatisch vom Videocontroller der Grafikkarte vorgenommen wird. Auf dem Portfolio-Bildschirm wird man deshalb nichts von dem sehen können, was das Programm in das Video-RAM hineinschreibt. Auch diese Pro-

blematik hat DIP erkannt: Durch eine weitere Funktion ($1F) des Interrupt $61 wird es ermöglicht, einen Screen-Refresh automatisch bei Auslösung eines NMI (Non-maskable Interrupt, wird durch einen Timer ausgelöst) und/oder bei der Betätigung der Tastatur durchführen zu lassen, wobei letztere Methode leider einen Haken hat, dazu aber später.

Wie auch der PC besitzt der Portfolio einen Timer, der regelmäßig einen Interrupt ($08) auslöst: Während im PC dieser jedoch 18,2mal in der Sekunde aufgerufen wird, ist dies im Portfolio wahlweise einmal in der Sekunde oder sogar nur alle 128 Sekunden (hierdurch wird Strom gespart!) der Fall. Neben diesem Interrupt kann der Portfolio-Timer aber auch noch einen sogenannten NMI auslösen; dies ist ein nichtmaskierbarer Interrupt, d.h., er kann nicht softwaremäßig unterdrückt werden. Durch diesen NMI kann schließlich ein Bildschirm-Refresh erzeugt werden. In der Praxis hat es natürlich keinen Sinn, einen Refresh nur alle 128 Sekunden auszuführen, so daß man zunächst den Timer so programmieren muß, daß der NMI jede Sekunde ausgelöst wird. Hierbei wird zwar erheblich mehr Strom verbraucht, aber schließlich ist es unzumutbar, jedesmal über zwei Minuten warten zu müssen, bis die neuen Bildschirminformationen sichtbar werden!

Für den Timer steht die Funktion $1E des erweiterten BIOS-Interrupt $61 zur Verfügung. Mit ihr kann man die Frequenz neu setzen (Register AL =1) oder auch nur auslesen (Register AL =0). Im ersten Fall wird im BX-Register die neue Frequenz übergeben:

BX = 0: Interrupt alle 128 Sekunden
BX = 1: Interrupt jede Sekunde

Beim Auslesen wird ein entsprechender Wert im BX-Register zurückgegeben.

Beispiel:
AH = $1E, AL = $01, BX = 1, Aufruf INT $61
Der Timer löst jede Sekunde einen Interrupt aus.

Mit der Interrupt-Funktion $1F kann man nun festlegen, wann ein Screen-Refresh ausgelöst werden soll. Dazu wird im DX-Register ein entsprechender Wert übergeben:

DX = $0000: Kein automatischer Refresh
DX = $0001: Refresh durch Tastatur
DX = $0100: Refresh durch Timer
DX = $0101: Refresh durch Timer und Tastatur

Wenn im AL-Register eine 1 übergeben wird, wird der neue Modus gesetzt, bei AL = 0 wird der aktuelle Modus im DX-Register zurückgegeben.

Beispiel:
AH = $1F,AL = $01, DX = $0100, Aufruf INT $61
Bei jedem Timer-Interrupt wird ein Screen-Refresh vorgenommen.

Nun zu dem oben erwähnten Haken mit der Tastatur: Schlauerweise wurde die Funktion so eingerichtet, daß der Refresh nicht sofort beim Druck einer Taste ausgelöst wird, sondern erst dann, wenn das Zeichen über den BIOS-Interrupt $16 ausgelesen wird. Warum ist dies sinnvoll? Nun, meistens ist es so, daß ein Programm den Bildschirm beschreibt und dann auf den Druck einer Taste wartet. Wenn nun erst das halbe Video-RAM vollgeschrieben wurde und eine Taste gedrückt wird, hätte dies einen Screen-Refresh zur Folge. Da das Zeichen aber im Tastaturpuffer gespeichert wird, kann es sofort ausgelesen werden, wobei kein zweiter Refresh ausgelöst würde. Damit hätte man keine Möglichkeit mehr, die zweite Hälfte der Bildschirmausgabe sichtbar zu machen. Man sieht also, daß es grundsätzlich sinnvoll ist, den Refresh erst dann auszulösen, wenn ein Zeichen aus dem Tastaturpuffer ausgelesen wird. Das einzige Problem tritt dann auf, wenn das Programm sofort nach diesem Auslesen den Bildschirm löscht: Man hat dann keine Chance mehr, den Bildschirminhalt in dieser kurzen Zeit zu lesen. Da dieser Fall gar nicht so selten auftritt, kann ich Ihnen nur empfehlen, den Refresh durch den Timer (1 Interrupt pro Sekunde) ausführen zu lassen, auch wenn Ihr Batteriebedarf in die Höhe schnellen wird. Alle anderen Möglichkeiten sind in der Praxis aber mehr oder weniger problematisch. Im Haus sollte ohnehin das umweltfreundliche Netzgerät zum Einsatz kommen.

Als letzten Punkt der Bildschirmprogrammierung möchte ich auf die Darstellungsart des Cursors eingehen. Im Gegensatz zu normalen Zeichen kann dessen Aussehen nämlich durchaus modifiziert werden. Auch hierfür existiert eine Funktion ($0F) des Interrupt $61. Insgesamt bestehen drei Möglichkeiten: Wird im Register AL eine 1 übergeben, muß im Register BL das Aussehen des Cursors übergeben werden:

BL = 0: Kein Cursor sichtbar
BL = 1: Cursor wird durch Unterstrich dargestellt
BL = 2: Cursor wird durch Block dargestellt

Bei dem Wert 2 im AL-Register wird der Cursor automatisch dem *Num-Lock*-Status angepaßt: Ist kein *NumLock* aktiviert, wird er als Block dargestellt, bei aktiviertem *NumLock* hingegen als Unterstrich. In diesem Fall

spielt der Inhalt des BL-Registers keine Rolle. Durch den Wert 0 im AL-Register schließlich wird der aktuelle Cursormode ermittelt und im Register BL zurückgegeben.

Beispiel:
AH = $0F, AL = $01, BL = $01, Aufruf INT $61
Der Cursor wird als Unterstrich dargestellt.

Zusammenfassend kann man sagen, daß der LCD-Controller doch etwas höhere Ansprüche an den Programmierer stellt, als er das vielleicht vom PC gewohnt ist. Da alle wesentlichen Funktionen jedoch über den Interrupt $61 abgedeckt sind, dürften eigentlich keine großen Probleme auftreten, obwohl man sich z.B. an die Screen-Refreshs erst gewöhnen muß. Massive Probleme können mit PC-Software entstehen, wenn diese versucht, direkt auf die (im Portfolio nicht vorhandene) Video-Hardware des PC zuzugreifen. So führt z.B. eine Benutzung der CRT-Unit von Turbo Pascal zum sofortigen Systemabsturz. In Kapitel 2 wird ausführlich auf die praktische Bildschirm-Programmierung eingegangen.

Was noch verbleibt, ist der Grafikmodus, dessen Programmierung gegenüber dem Textmodus fast als trivial zu bezeichnen ist. Er besitzt nur eine Auflösung (240x64 Punkte), wobei Grafikpunkte ganz normal über die Funktion $0C des BIOS-Interrupt $10 gesetzt werden können. Welcher Farbwert im Register AL für die Zeichenfarbe übergeben wird, ist so ziemlich egal, nur der Wert 0 veranlaßt, daß der Punkt in der Hintergrundfarbe gesetzt, also gelöscht wird. Ob Sie zum Setzen eines Punktes aber den Wert 1 oder 15 übergeben, spielt keine Rolle. Auch zum Auslesen eines Pixels kann die hierfür vorgesehende Funktion $0D des Interrupt $10 verwendet werden. Für die Verfechter der Methode des direkten Beschreibens des Video-RAM sei gesagt, daß der Screen-Refresh (INT $61, Funktion $12) im Grafimodus so lange dauert, daß es sich im Gegensatz zum PC beim Portfolio überhaupt nicht lohnt, das Video-RAM direkt zu beschreiben. (Es gibt eine interne Funktion, die nur jeweils ein Grafikbyte »refresht«, da diese aber nicht dokumentiert ist, sollte man auf sie verzichten, zumal die Geschwindigkeitsvorteile wirklich nur minimal sind.) Was sich aber durchaus lohnt, ist das direkte Auszulesen des Video-RAM. Diese Funktion wird z.B. von einer *Fill*-Routine häufig benötigt, wodurch sich insgesamt spürbare Vorteile ergeben. Deshalb möchte ich noch kurz auf den Aufbau des Video-RAM im Grafikmodus eingehen.

Wie im Textmodus beginnt es bei $B000:$0000. Die Grafikbytes sind zeilenweise von links nach rechts abgespeichert, wobei immer acht Bildpunkte in einem Byte abgelegt sind (ein Punkt entspricht einem Bit). Falls

das entsprechende Bit gesetzt ist, heißt dies, daß der Punkt in der Vordergrundfarbe gesetzt wird, bei gelöschtem Bit wird der Punkt unsichtbar in der Hintergrundfarbe dargestellt. Pro Zeile werden somit 240/8= 30 Grafikbyte benötigt. Innerhalb eines Grafikbytes wird der am weitesten links liegende Punkt durch das höchstwertige Bit (7) repräsentiert, der am weitesten rechts liegende durch das niederwertigste Bit (0). Insgesamt werden 240x64/8 = 1920 Byte benötigt. Da der Rest des Video-RAM im Grafikmodus unbenutzt bleibt, kann man eigene Daten hineinschreiben, wie es in Kapitel 2 praktiziert wird.

	x-Koordinate					
y-Koordinate					22222222	22222222
		111111	11112222		22222233	33333333
	01234567	89012345	67890123	.	45678901	23456789
0	$0000	$0001	$0002	.	$001C	$001D
1	$001E	$001F	$0020	.	$003A	$003B
2	$003C	$003D	$003E	.	$0058	$0059
3	$005A	$005B	$005C	.	$0076	$0077
4	$0078	$0079	$007A	.	$0094	$0095
..				.		
60	$0708	$0709	$070A	.	$0724	$0725
61	$0726	$0727	$0728	.	$0742	$0743
62	$0744	$0745	$0746	.	$0760	$0761
63	$0762	$0763	$0764	.	$077E	$077F

Bild 1.5: Aufbau des Video-RAM im Grafikmodus

Als äußerst nachteilhaft muß man die Tatsache ansehen, daß auf das Character-ROM (das ist der Baustein, in dem das Aussehen der einzelnen Zeichen abgelegt ist) nur vom LCD-Controller, nicht aber vom Prozessor zugegriffen werden kann, da es sich nicht in dessen Adreßraum befindet. Damit hat nämlich das BIOS keine Möglichkeit, diese Informationen auszulesen, weshalb es auch nicht möglich ist, irgendwelche Zeichen mit Hilfe der BIOS-Funktionen (z.B. INT $10, Funktion $0A) im Grafikmodus auf dem Bildschirm auszugeben. Beim PC hingegen liegen die Daten der ersten 128 ASCII-Codes bereits im ROM bereit (Grund hierfür ist die CGA-Karte, die kein eigenes Character-ROM besitzt), die Daten der zweiten 128 Zeichen können mit Hilfe des DOS-Befehls GRAFTABL in einer Tabelle initialisiert werden, deren Anfangsadresse in der Interrupt-Vektor-Tabelle

ab der Position $0000:$007C (der Interrupt $1F wird nicht benutzt!) zur Verfügung steht. Damit können die BIOS-Funktionen für Zeichen- oder Textausgabe diese Daten auslesen und Punkt für Punkt auf den Bildschirm übertragen. Dies alles trifft für den Portfolio nicht zu. Die einzige Möglichkeit, Grafiken auch beschriften zu können, besteht darin, eine eigene Zeichen-Code-Tabelle aufzubauen und deren Daten mit einem selbstgeschriebenen Programm Punkt für Punkt auszulesen und darzustellen (siehe hierzu Kapitel 2, Grafikprogrammierung).

Auch hier gestatten Sie mir eine Schlußbemerkung: Der Portfolio-Grafikmodus ist inkompatibel zu allen beim PC bekannten Modi. Deshalb ist es nicht möglich, irgendein für den PC entwickeltes Grafikprogramm auf dem Portfolio ablaufen zu lassen. Hier hilft nur die Devise »Selbst ist der Mann/die Frau/der Programmierer«. Leider ist es aus diesem Grund auch nicht möglich, ein für den Portfolio konzipiertes Grafikprogramm auf dem PC zu testen, da dieser mit dem Video-Modus $0A nichts anfangen kann. (Die meisten Rechner stürzen ab, wenn Sie den BIOS-Interrupt $10 mit diesem Parameter aufrufen!) Hier bietet sich die Lösung an, die Grafikroutinen völlig aufrufkompatibel zu entsprechenden PC-Routinen zu machen, um dann einfach die Grafikpakete wechseln zu können, ohne an dem Programm selbst eine Anweisung umschreiben zu müssen. Es ist deshalb auch sinnvoll, alle Portfolio-spezifischen Interrupt-Aufrufe direkt vom Grafikpaket und nicht vom Hauptprogramm ausführen zu lassen.

1.4 Die RAM-Speicherkarten (CCMs)

Aufgrund seiner geringen Abmessungen konnte in den Portfolio im Gegensatz zu »vollwertigen« Laptops weder ein Diskettenlaufwerk noch eine Festplatte integriert werden. Da ein Rechner ohne Datenspeicher jedoch wertlos ist, spendierten ihm seine Entwickler einen Einschub für ein sogenanntes Credit-Card-Memory (CCM), fälschlicherweise oft mit dem Ausdruck »Magnetspeicherkarte« bezeichnet. In Wirklichkeit haben diese Karten nämlich (im Gegensatz zu Disketten) nichts, aber auch gar nichts mit Magnetismus zu tun, es handelt sich schlicht und ergreifend um ein ganz normales RAM im Kartenformat, das den Voteil aufweist, batteriegepuffert zu sein, weshalb die Informationen auch bei Entfernen der Karten aus dem Rechner nicht verlorengehen. Ein CCM ist also eine RAM-Disk, wie man sie beim PC z.B. mit dem Gerätetreiber RAMDRIVE.SYS einrichten kann.

In diesem RAM können nun Programme und Dateien im DOS-Format abgelegt werden, wobei ich auf die Dateiverwaltung des DOS gleich zu sprechen komme. Aus Abschnitt 1.1 wissen Sie, daß das CCM in den Adreßraum des Portfolio eingeblendet werden kann. Ich habe mir daher die Frage gestellt, was einen daran hindern kann, ein CCM als ganz normale RAM-Erweiterung für den Hauptspeicher zu benutzen. Die Antwort: nichts. Wenn Sie das RAM eingeblendet haben, können Sie die Speicherzellen ganz normal wie jedes RAM adressieren und brauchen sich überhaupt nicht darum zu kümmern, daß diese Dinger ursprünglich als Dateispeicher vorgesehen waren. Zwar wird der EXEC-Loader kein Programm in diesen Bereich ab $C000:$0000 laden können, als Datenspeicher kann man ihn aber allemal benutzen, zumal es Hochsprachen wie Turbo Pascal möglich machen, Variablen an fest vorgegebenen Adressen anzulegen. Obwohl ich auf die Programmierung der Karten als Variablenspeicher erst in Kapitel 2 eingehe, möchte ich Ihnen schon einen kleinen Vorgeschmack geben. Stellen Sie sich vor, Sie wollen eine Adreßverwaltung programmieren, deren Programmcode ca. 90 Kbyte umfaßt. Der Hauptspeicher bietet dann nur noch maximal 10 Kbyte Datenspeicher an, wobei die eingebauten Applikationen nicht mehr aufgerufen werden könnten. In Turbo Pascal würde man zunächst einen Record definieren, der eine Adresse aufnehmen kann:

```
TYPE
adresse = record
 name : string[20];
 vorname : string[20];
 strasse : string[20];
 plz : word;
 wohnort : string[15];
 telefon : word;
end;
```

Dieser Record benötigt 83 Byte, da beim String noch 1 Byte für die Längenangabe hinzugerechnet werden muß (21+21+21+2+16+2). Mit 10 Kbyte Speicher könnten somit 120 Adressen gespeichert werden, was für eine so umfangreiche Adreßverwaltung nicht gerade viel ist. Eine Variable könnte somit wie folgt initialisiert werden:

```
VAR
speicher : array[1..120] of adresse;
```

Falls Sie nun z.B. ein 64-Kbyte-CCM besitzen, könnten Sie diesen Speicher voll in Ihr Programm integrieren, indem Sie das Variablenarray durch

Angabe einer Absolutadresse zwangsweise in das CCM-RAM verlagern. Damit können nicht wie bisher nur 120 Datensätze gespeichert werden, sondern deren 65536/83=789, wobei die 10 Kbyte Hauptspeicher noch nicht berücksichtigt werden. Durch die Initialisierung eines zweiten Array kann die Gesamtzahl der Datensätze auf 789+120=909 gesteigert werden:

```
VAR
speicher1 : array[1..789] of adresse ABSOLUT $C000:$0000;
speicher2 : array[1..120] of adresse;
```

Durch die Angabe *ABSOLUT* wird Turbo Pascal angewiesen, die Daten ab der Adresse $C000:$0000 abzulegen, also im Adreßbereich des CCM. Sie können nun berechtigterweise fragen, was dieser Umstand soll, wo es doch eine »normale« RAM-Erweiterung für den Portfolio gibt. Nun, diese hat gegenüber den CCMs drei entscheidende Nachteile:

1. Durch eine RAM-Erweiterung wird der Rechner wesentlich vergrößert, so daß von Taschenformat keine Rede mehr sein kann. Ein CCM dagegen benötigt keinen zusätzlichen Platz.

2. Eine RAM-Erweiterung weist einen enormen Strombedarf auf, wodurch die Lebensdauer der Batterien wesentlich verkürzt wird. Ein CCM benötigt praktisch keinen Strom, der den Portfolio belasten würde.

3. Die Informationen einer RAM-Erweiterung sind verloren, sobald das Programm beendet wird, da innerhalb des Hauptspeichers Programme an keine feste Adresse geladen werden. Jegliche Art von Daten müssen somit in Dateien abgelegt werden, wofür nur die interne RAM-Disk oder CCMs in Frage kommen. Die Daten eines CCM sind jedoch batteriegepuffert und liegen an einer festen Adresse, so daß ein Programm jederzeit auf sie zugreifen kann, ohne eine Datei ausgelesen haben zu müssen.

Der einzige Nachteil eines CCM besteht darin, daß ihr RAM nicht als Erweiterung für Programmcode benutzt werden kann, der nicht in den Hauptspeicher paßt. Da ich aber nicht glaube, daß für den Portfolio jemals Programme angeboten werden, deren Code größer als 100 Kbyte ist, kann man hier wohl getrost von einem eher theoretischen Nachteil sprechen. Auf jeden Fall hoffe ich, Ihnen etwas Appetit auf diese zugegeben unkonventionelle Verwendung der CCMs gemacht zu haben. Um ein CCM in den Adreßraum des Portfolio einzublenden, kann die Funktion $24 des Interrupt $61 benutzt werden. Dazu wird im AL-Register eine 1 übergeben (Status setzen, AL = 0 würde aktuellen Status holen), im DL-Register wird der neue Status übergeben:

DL = 0: Applikations-ROM wird in Adreßraum eingeblendet
DL = 1: CCM A: wird in Adreßraum eingeblendet
DL = 2: CCM B: wird in Adreßraum eingeblendet
DL = 3: Externes ROM wird in Adreßraum eingeblendet

Im Register DH wird der Status der CCMs übergeben:

DH = 0: Keine CCMs ansprechbar
DH = 1: CCM A: ansprechbar
DH = 2: CCM B: ansprechbar

Natürlich sind nicht alle Kombinationen von DL und DH sinnvoll. So wäre es z.B. nicht möglich, in DL durch die 1 das CCM A: einblenden zu wollen, es aber andererseits durch DH = 0 nicht zugänglich zu machen.

Nach dem Interrupt-Aufruf wird im Carry-Flag signalisiert, ob die Aktion erfolgreich war: Durch ein gelöschtes Flag wird der fehlerlose Ablauf bestätigt, während ein gesetztes Flag einen Fehler bzw. einen illegalen Parameter in DL und/oder DH anzeigt.

Beispiel:
AH = $24, AL = $01, DL = $01, DH = $01, Aufruf INT $61
Das CCM A: wird in den Adreßraum eingeblendet.

Bevor man ein CCM einblendet, sollte man überprüfen, ob überhaupt eine RAM-Karte eingesteckt ist. Dazu kann die Interrupt-Funktion $0B verwendet werden. Dazu wird im Register AL die Nummer des CCM übergeben (0= CCM A:, 1= RAM-Disk B:), das gelösche Carry-Flag nach Rückkehr aus der Interrupt-Service-Routine signalisiert, daß die Karte eingesteckt (bei CCM A:) bzw. daß eine RAM-Disk eingerichtet ist (bei CCM B:), während bei gesetztem Carry-Flag ein Fehler aufgetreten ist. Der Fehlercode entspricht dem des Interrupt $13, der für den DOS-Zugriff auf Massenspeicher zuständig ist (s.u.), und wird im Register AH übergeben.

Beispiel:
AH = $0B, AL = $00, Aufruf INT $61
Test, ob RAM-Karte eingelegt wurde.

Wichtig ist, daß man einen Aufruf der Applikationen verhindert, wenn man ein CCM in den Adreßraum eingeblendet hat, da sonst eine Programm-Routine zur Ausführung kommt, die im CCM gar nicht vorhanden ist. In diesem Fall kann man mit Sicherheit von einem Systemabsturz ausgehen.

Wie gesagt sind die RAM-Karten eigentlich als Ersatz für fehlende Massenspeicher gedacht, wofür sie wahrscheinlich auch häufiger benutzt werden als für Speichererweiterungen. Die Dateiverwaltung wird von DOS weitgehend geräteunabhängig vorgenommen, erst der unmittelbare Zugriff auf die Hardware erfolgt durch die einzelnen Gerätetreiber. Man darf sich deshalb nicht wundern, wenn der Aufbau des CCM aus der Sicht von DOS dem einer Diskette entspricht, obwohl man RAM-Speicher nicht mit einer Magnetscheibe vergleichen kann.

Als oberste Ebene sieht DOS das *Volume* an. Hierbei handelt es sich um einen Massenspeicher (oder einen Teil dessen) an sich, der über einen Namen (A:, B:, C: etc.) angesprochen wird. Es handelt sich dabei um logische Einheiten, bekanntlich kann ja eine Festplatte in mehrere DOS-Partitionen unterteilt werden. Jedes *Volume* enthält eine bestimmte Anzahl von Spuren (Tracks), wobei man sich dies bei Disketten am anschaulichsten als über die Scheibe verteilte Ringe vorstellen kann, auf die der Schreib-Lese-Kopf durch schrittweise Bewegung positioniert werden kann. Jeder Track wird nochmals in Sektoren unterteilt, die dann die Daten aufnehmen. Der Vorteil dieser Aufsplitterung eines *Volume* in kleine Abschnitte besteht darin, daß man insbesondere bei Disketten das zur Verfügung stehende Gesamtvolumen wesentlich besser ausnutzen kann als dies bei größeren Einheiten, wie z.B. Tracks, der Fall wäre: Stellen Sie sich vor, ein Track könnte 10000 Datenbyte aufnehmen. Wenn nun eine Datei mit nur 1000 Byte gespeichert würde, könnten die anderen 9000 Byte nicht mehr benutzt werden, da ja jeder Track nur eine Anfangsmarkierung aufweist. Wäre der Track jedoch z.B. in zehn Sektoren mit jeweils 1000 Byte unterteilt, könnte das Programm in einem Sektor untergebracht werden, während die anderen neun Sektoren noch zur freien Verfügung ständen. (Jeder Sektor besitzt eine eigene Anfangsmarkierung auf der Diskette!) In der Praxis gehen auf einer Diskette natürlich einige Bytes durch die Sektormarkierungen verloren, so daß die obige Rechnung nicht ganz aufgeht.

Bei einer RAM-Disk könnte man auf Tracks und Sektoren natürlich verzichten, da hier lediglich Adressen angesprochen werden und keinerlei Markierungen (wie sie der Schreib-Lese-Kopf eines Diskettenlaufwerks benötigt) erforderlich sind. Um die Verwaltung von DOS übernehmen lassen zu können, ist eine Kompatibilität in dieser Richtung aber erforderlich. Selbstverständlich könnte man theoretisch ein eigenes Betriebssystem für RAM-Disks entwickeln, das mit Sicherheit effektiver und schneller wäre als DOS. Dafür könnte es aber auch ausschließlich für RAM-Disks verwendet werden, während DOS auch Disketten, Festplatten und CD-ROMs verwalten kann.

Das Problem für DOS besteht nun darin, daß sich Zugriffe vom Programm aus immer auf Dateien und nicht auf Sektoren beziehen. Die Dateizugriffe müssen also über irgendeinen Algorithmus in Sektor-Zugriffe umgewandelt werden. Neben den Verzeichnissen existiert hierfür eine sogenannte FAT (File-Allocation-Table), in der die von einer Datei benutzten Sektoren abgelegt sind. Weiterhin existiert in jedem *Volume* ein Boot-Sektor, der die wesentlichen Informationen über den Massenspeicher (z.B. Anzahl der Leseköpfe, der Tracks, Sektoren etc.) enthält und in jedem Fall auf Track 0 zu finden ist (Sektor-Nummer 0), damit ihn DOS finden und auswerten kann.

Der Name Boot-Sektor ergibt sich daraus, daß beim PC das DOS nicht fest im Speicher verankert ist, sondern erst eingeladen (gebootet) werden muß. Dafür lädt das BIOS nach seiner eigenen Initialisierung den Boot-Sektor der eingelegten System-Diskette bzw. der Festplatte, deren Boot-Routine DOS dann einladen kann. Beim Portfolio braucht DOS nicht geladen zu werden, weshalb die Boot-Routine der CCMs lediglich eine Fehlermeldung ausgibt (s.u.). Im folgenden ist der Aufbau des Boot-Sektors in allgemeiner Form dargestellt:

Aufbau des Boot-Sektors eines Volume

Nr.	Adr.	Inhalt	Byte
1	$00	JMP-Befehl zur Boot-Routine	3
2	$03	Herstellername/DOS-Versionsnummer	8
3	$0B	Bytes pro Sektor	2
4	$0D	Sektoren pro Cluster	2
5	$0E	Anzahl reservierter Sektoren	2
6	$10	Anzahl File-Allocation-Tables (FAT)	1
7	$11	Maximale Einträge im Hauptverzeichnis	2
8	$13	Anzahl Sektoren im Volume	2
9	$15	Media-Discriptor	1
10	$16	Anzahl Sektoren pro FAT	2
11	$18	Anzahl Sektoren pro Track	2
12	$1A	Anzahl Schreib-Lese-Köpfe	2
13	$1C	Entfernung erster Sektor des Volume vom ersten Sektor des Massenspeicher	2
14	$1E	Bootroutine (bis $1FF)	482

Bild 1.6: Aufbau des Bootsektors eines Volume

Zunächst zu den Clustern: Grundsätzlich beschreibt DOS nicht direkt aufeinanderfolgende Sektoren, sondern immer den nächsten freien Sektor. Während bei einem relativ neuen *Volume* die Dateien auf wenig Tracks verteilt sind, wird nach einiger Zeit durch Löschen und Neubeschreiben eine Zerstückelung der Dateien über immer mehr Tracks unvermeidlich. Dadurch kann es im Extremfall dazu kommen, daß eine Datei auf jedem Track nur einen einzigen Sektor belegt. Bei RAM-Disks wäre dies unproblematisch, bei Festplatten aber müßte der Lesekopf auf jedem Track neu positioniert werden, wodurch eine extreme Verzögerung im Datenzugriff entstehen würde. Um hier Abhilfe zu schaffen, faßt DOS eine bestimmte Anzahl von direkt aufeinanderfolgenden Sektoren zu den sogenannten Clustern zusammen, die nun ihrerseits die kleinste mögliche Einheit in dem *Volume* bilden. Eine Datei, die z.B. 20 Sektoren belegt hätte, würde bei einer Zusammenfassung von jeweils vier Sektoren zu einem Cluster also fünf Cluster belegen, wodurch statt 20 nur noch maximal fünf Positionierungen des Lesekopfs notwendig wären. Es können immer nur Sektor-Zahlen zu einem Cluster zusammengefaßt werden, die eine Potenz von 2 bilden, also 1, 2, 4, 8, 16 usw. Leider wird durch dieses Verfahren der Anteil des brachliegenden Speichers aber drastisch erhöht: Würde ein Sektor z.B. 256 Byte enthalten und ein Programm mit 250 Byte abgespeichert, würden bei einzelnem Sektorzugriff nur 6 Byte verschwendet, während durch die Zusammenfassung von vier Sektoren zu einem Cluster auch die drei übrigen Sektoren des Clusters durch andere Dateien nicht mehr benutzt werden können, so daß 3x256+6=774 Byte unbenutzt sind. Bei Festplatten kann wie gesagt aufgrund der erforderlichen Geschwindigkeit auf die Zusammenfassung mehrerer Sektoren nicht verzichtet werden, zumal diese Massenspeicher so dimensioniert sind, daß es auf das eine oder andere verlorene Byte nicht ankommt. Anders sieht es z.B. bei den Disketten aus: Hier wird aufgrund der geringen Speicherkapazität keine Zusammenfassung vorgenommen (ein Sektor pro Cluster), wobei man sich überlegt hat, daß Disketten meistens nur zum Datentransfer benutzt werden, wobei es mehr auf das Fassungsvermögen als auf die Geschwindigkeit ankommt. Bei einer RAM-Disk schließlich spielt es natürlich überhaupt keine Rolle, auf welchen Sektor zugeriffen wird. Da keine mechanische Bewegung eines Lesekopfes notwendig ist, wird (natürlich) nur 1 Sektor pro Cluster verwendet. Man sieht, wie flexibel DOS mit dieser Methode ist: Optimierung der Zugriffszeit (Festplatte) oder der Speicherausnutzung (Diskette, RAM-Disk) mit ein und demselben Verfahren!

Mit den reservierten Sektoren sind der Boot-Sektor sowie eventuell weitere Sektoren gemeint, die für die Speicherung der Dateien nicht zur Ver-

fügung stehen. Im Normalfall ist nur der Boot-Sektor reserviert, falls sich aber z.B. die Boot-Routine noch über weitere Sektoren erstreckt, müssen auch diese als reserviert gekennzeichnet werden, damit sie von DOS nicht mit Daten überschrieben werden.

Nun zu der FAT-Anzahl. Wie schon oben erwähnt, dient die FAT dazu, DOS mitzuteilen, auf welchen Clustern welche Dateien gespeichert sind. Wenn die FAT aus irgendeinem Grund zerstört wird, hat man daher keine Chance mehr, auf irgendwelche Dateien zuzugreifen, der gesamte *Volume*-Inhalt ist verloren. DOS als Profi-Betriebssystem bietet deshalb die Möglichkeit, Kopien der FAT anzulegen, die zwar Speicherplatz kosten, der für Daten nicht mehr zur Verfügung steht, dafür aber die Daten-Sicherheit wesentlich erhöhen.

Beim Media-Descriptor handelt es sich um einen Eintrag, der das Gerät angibt, mit dem das Speichermedium (z.B. Diskette) verarbeitet werden kann. Dabei sind beim PC folgende Einträge möglich:

Media-Descriptor bei Speichermedien

Medium	Seiten	Größe	Tracks	Sek/Track	M-D	Kbyte
Diskette	2	3½	80	18	$F0	1440
Harddisk	?	??	??	??	$F8	????
Diskette	2	5	80	15	$F9	1200
Diskette	2	3½	80	9	$F9	720
Diskette	1	5	40	9	$FC	180
Diskette	2	5	40	9	$FD	360
Diskette	1	5	40	8	$FE	160
Diskette	2	5	40	8	$FF	320

Bild 1.7: Media-Descriptor bei diversen Speichermedien

Da ein Massenspeicher in mehrere *Volumes* unterteilt werden kann (Festplatte), ist es nicht zwingend, daß der erste Sektor des *Volume* auch der erste Sektor des Massenspeichers ist. (Dies ist nur beim ersten *Volume* des Massenspeichers der Fall!) Der Abstand des ersten Sektors ist deshalb im Boot-Sektor ab der Adresse $1C vermerkt.

Nach dieser allgemeinen Einführung möchte ich Ihnen nun die spezifischen Einträge des DIP-DOS in die Boot-Sektoren der CCMs vorstellen:

Nr.	32 Kbyte	64 Kbyte	128 Kbyte
1	JMP $0024	JMP $0024	JMP $0024
2	DIP 2.0	DIP 2.0	DIP 2.0
3	128	256	512
4	1	1	1
5	1	1	1
6	1	1	1
7	32	64	128
8	256	256	256
9	255	255	255
10	3	2	1
11	8	8	8
12	2	2	2
13	0	0	0
14	ab $0024 siehe unten		

Bild 1.8: Belegung des Bootsektors bei CCMs

Man sieht, daß DIP den einfachen Weg gegangen ist, die unterschiedlichen Speichergrößen über unterschiedliche Sektorgrößen zu realisieren, damit die Anzahl der Sektoren und damit die der FAT-Einträge konstant bleibt. Durch die kleine Sektorgröße bedingt kann der paradoxe Fall auftreten, daß auf der 32-Kbyte-Karte genauso viele Dateien gespeichert werden können wie auf der 128-Kbyte-Karte, nämlich dann, wenn die Dateien weniger als 128 Byte umfassen. Optimaler wäre der Fall gewesen, wenn bei der 64-Kbyte- und 128-Kbyte-Karte die Sektorgröße ebenfalls 128 Byte betragen hätte und dafür die Anzahl der Sektoren gegenüber der 32-Kbyte-Karte verdoppelt bzw. vervierfacht worden wäre, weil dann eine bessere Ausnutzung des RAM gegeben gewesen wäre. An Hand des Media-Descriptors erkennt man, daß DOS die CCMs wie 320-Kbyte-Disketten verwaltet, wobei die Tracknummern allerdings nur von 0 bis 15 (auf jeder »Seite« 16 Tracks ergibt insgesamt 32 Tracks) und die Sektornummern von 1 bis 8 laufen. Die Anzahl der »Seiten« darf man natürlich nicht so ernst nehmen, sie werden jedoch zur Berechnung des Track benötigt (Track= Seitex16+Tracknummer). Die Pseudo-Boot-Routine (der Portfolio wird über das ROM gebootet!) sieht bei allen drei Karten wie folgt aus:

```
0024 B80700        MOV AX,0007    ;Textmodus einschalten
0027 CD10          INT 10
0029 B80113        MOV AX,1301    ;Zeichenkette ausgeben
002C BB0700        MOV BX,0007    ;Bildschirmseite 0
002F 33D2          XOR DX,DX      ;Zeile=Spalte=0 setzen
0031 0E            PUSH CS        ;Segmentadresse = Codesegment
0032 07            POP ES
0033 BD4300        MOV BP,0043    ;Offsetadresse
0036 90            NOP
0037 B93900        MOV CX,0039    ;Anzahl der auszugebenen Zeichen
003A 90            NOP
003B CD10          INT 10         ;INT $10, Funktion $13 aufrufen
003D B400          MOV AH,00
003F CD16          INT 16         ;Auf Tastendruck warten
0041 CD19          INT 19         ;Rechner booten
0043  4E 6F 6E 2D 53 79 73 74 65 6D 20 64 69 73 6B 20  Non-System disk
0053  6F 72 20 64 69 73 6B 20 65 72 72 6F 72 2E 0D 0A  or disk error
0063  52 65 70 6C 61 63 65 20 61 6E 64 20 70 72 65 73  Replace and press
0073  73 20 61 6E 79 20 6B 65 79 00 00 00 00 00 00 00  any key
```

Wenn Sie ein CCM einschieben und die Tasten [Alt]+[Strg]+[Entf] drücken, sehen Sie, daß diese Boot-Routine nicht durchlaufen wird. Zwar gibt es eine Möglichkeit, über ein CCM in den Boot-Vorgang einzugreifen, dazu muß der Boot-Sektor jedoch ein ganz bestimmtes Format aufweisen. Nachdem der Boot-Sektor besprochen wurde, können wir uns dem Aufbau der FAT zuwenden.

Die FAT beginnt direkt hinter dem letzten reservierten Sektor, wobei bei mehreren vorhandenen FATs die Kopien ihrerseits direkt an die Original-FAT anschließen. Jeder FAT-Eintrag belegt 12 Bit, so daß jeweils zwei Einträge 3 Byte in Anspruch nehmen. In der MS-DOS-Version 3 werden übrigens 16 Bit pro Eintrag reserviert, wenn mehr als 4096 Cluster vorhanden sind, was nicht verwunderlich ist, da man mit 12 Bit gerade 2^{12}=4096 Zahlen darstellen kann. Um die FAT-Einträge richtig interpretieren zu können, liest DOS zunächst die Gesamtanzahl der Sektoren aus dem Boot-Sektor aus und teilt diese durch die Anzahl der Sektoren pro Cluster. Die sich ergebende Cluster-Zahl bestimmt, ob in der FAT 12 oder 16 Bit pro Eintrag verwendet werden.

Beim Portfolio interessiert dies allerdings nicht, da das DIP-DOS kompatibel zu MS-DOS Version 2 ist, so daß in jedem Fall 12 Bit verwendet werden (eigentlich bräuchte man nur 8 Bit, da nur 256 Sektoren existieren). Die ersten beiden FAT-Einträge sind reserviert, wobei nur ein Byte benutzt wird, nämlich das erste, das eine Kopie des Media-Descriptors enthält. Die übrigen 2 Byte enthalten Füllwerte ($FF). Um zu verstehen, wie die FAT genau funktioniert, muß man wissen, daß in jedem

Verzeichnis-Eintrag die Cluster-Nummer des ersten Datenblocks der Datei gespeichert ist. In der FAT findet sich die zu einem Cluster korrespondierende Bitposition, indem man die Cluster-Nummer mit 12 multipliziert. Da die ersten beiden Einträge reserviert sind, beginnt DOS daher mit der Cluster-Nummer 2, die Nummern 0 und 1 werden nicht benutzt. Der FAT-Eintrag bietet nun weitere Informationen: Er kann im Normalfall entweder die Cluster-Nummer des nächsten Datenblocks der Datei beinhalten, so daß sich an Hand der Einträge eine Kette der benutzten Cluster ergibt oder eine Endemarkierung, die anzeigt, daß der entsprechende Cluster von der Datei zwar belegt wird, jedoch keine weiteren Cluster mehr folgen. Damit hat DOS mit der FAT zwei Fliegen mit einer Klappe geschlagen: Zum einen kann es feststellen, ob der entsprechende Cluster überhaupt belegt ist, zum anderen kann es aus seinem Inhalt unmittelbar auf den folgenden schließen. Diese Methode ist wesentlich effektiver als ein anderes gebräuchliches Verfahren, bei dem der Folgesektor innerhalb der eigentlichen Datenblöcke abgelegt ist und der Block nur durch 1 Bit gekennzeichnet wird (1=Block belegt, 0=Block ist unbelegt). Insgesamt kann ein Cluster folgende Einträge aufweisen:

Eintrag	Bedeutung
$000	Cluster ist unbelegt
$FF0 – $FF6	Cluster ist reserviert
$FF7	Cluster ist zerstört (unbelegbar)
$FF8 – $FFF	Letzter Cluster einer Datei
$???	Folge-Cluster einer Datei

Bild 1.9: Mögliche Cluster-Einträge in FAT

Bevor ich Ihnen an Hand eines konkreten Beispiels die Belegung der FAT demonstriere, möchte ich noch auf den Aufbau des Hauptverzeichnisses eingehen, das direkt an die letzte Kopie der FAT anschließt (soweit eine oder mehrere Kopien vorhanden sind). Die maximale Anzahl von Einträgen ist aus dem Boot-Sektor ersichtlich, jeder Eintrag belegt 32 Byte:

Aufbau eines Verzeichnis-Eintrags			
Nr.	**Adr.**	**Inhalt**	**Byte**
1	$00	Dateiname (mit Leerzeichen gefüllt)	8
2	$08	Extension (mit Leerzeichen gefüllt)	3
3	$0B	Attribut-Byte	1
4	$0C	Interne Verwendung	10
5	$16	Uhrzeit der letzten Veränderung	2
6	$18	Datum der letzten Veränderung	2
7	$1A	Erster Cluster der Datei	2
8	$1C	Dateigröße in Byte	4

Bild 1.10: Aufbau eines Verzeichnis-Eintrags

Der Dateiname sowie die Dateierweiterung (Extension) werden bei nicht voller Belegung der zulässigen Länge mit Leerzeichen aufgefüllt, der Punkt zwischen Dateiname und Extension wird nicht mit abgespeichert. Falls eine Datei gelöscht wird, wird einfach das erste Zeichen des Dateinamens auf den Wert $E5 gesetzt. Was aber passiert, falls eine Datei genau mit diesem ASCII-Code beginnt? Nun, in diesem Fall wird der erste Zeichencode zu einer $05 umgewandelt (das dem Code $05 entsprechende Zeichen wird nach Ansicht der DOS-Entwickler nicht für Dateinamen verwendet), was DOS dann als $E5 interpretiert. Man hätte die Sache auch einfacher haben können, indem man die $05 gleich als Kennzeichnung für eine gelöschte Datei verwendet hätte. Eine $00 signalisiert schließlich, daß keine weiteren Einträge im Verzeichnis folgen.

Im Attribut-Byte werden bestimmte Eigenschaften des Eintrags abgelegt:

Bit 0 gesetzt: Datei darf nicht beschrieben werden (Read-Only)
Bit 1 gesetzt: Versteckte Datei (Hidden)
Bit 2 gesetzt: Datei ist System-Datei
Bit 3 gesetzt: Eintrag ist Volume-Name
Bit 4 gesetzt: Eintrag ist Unterverzeichnis
Bit 5 gesetzt: Archivierungsbit
Bit 6–7: Unbenutzt

Durch Kombination von gesetzten Bits können die Dateien auch mehrere der aufgeführten Eigenschaften besitzen. Falls beim Formatieren eines *Volume* ein Name angegeben wird, wird Bit 3 gesetzt, so daß dieser Name

bei DOS-Befehlen wie DIR, VOL etc. angezeigt wird. Die anderen Bytes dieses Eintrags sind dann unwesentlich. Durch ein gesetztes viertes Bit wird signalisiert, daß es sich um ein Unterverzeichnis handelt.

Für ein Unterverzeichnis wird ein ganz normaler Cluster reserviert, indem die Einträge in der gleichen Weise vorgenommen werden wie im Hauptverzeichnis. Im Eintrag des Hauptverzeichnisses werden die folgenden Informationen vermerkt: Der Name des Unterverzeichnisses im Dateinamen- und Extension-Feld, die Cluster-Nummer des ersten reservierten Clusters für das Unterverzeichnis sowie Datum und Uhrzeit seiner Erstellung. Das Dateilängen-Byte besitzt den Wert 0. Der zu dem Unterverzeichnis-Cluster korrespondierende FAT-Eintrag enthält die Nummer des Folge-Clusters, falls ein Cluster nicht für alle Einträge des Unterverzeichnisses ausreicht. Man kann also sagen, daß ein Unterverzeichnis wie eine normale Datei organisiert ist. Im Gegensatz dazu steht das Hauptverzeichnis, dessen Cluster logisch aufeinanderfolgen und dessen Größe durch die maximal zulässige Anzahl der Einträge begrenzt ist. Ein Unterverzeichnis kann dagegen theoretisch beliebig viele Einträge enthalten, wenn genug Cluster zur Verfügung stehen. Bei der Einrichtung eines Unterverzeichnisses werden gleich die zwei bekannten Dateien ».« und »..« angelegt, die nicht einzeln gelöscht werden können und erst bei der Löschung des gesamten Unterverzeichnisses entfernt werden. Die Datei ».« deutet dabei auf das aktuelle Unterverzeichnis, enthält also die gleiche Cluster-Nummer wie der Eintrag des Unterverzeichnisses im übergeordneteten Verzeichnis. Die Datei »..« hingegen deutet auf die Cluster-Nummer des übergeordneten Verzeichnisses, so daß DOS den Weg des Abstiegs zurückverfolgen kann. Falls das übergeordnete Verzeichnis das Hauptverzeichnis ist, enthält das Cluster-Feld den Wert 0.

Das Archivierungsbit wird bei jedem Schreibzugriff auf eine Datei gesetzt. Falls nun ein Daten-Sicherungsprogramm (z.B. BACKUP) das entsprechende *Volume* abarbeitet, setzt es das Archivierungsbit auf 0. Bei einem erneuten Durchlauf kann es nun feststellen, ob die Datei seit dem letzten Durchlauf modifiziert wurde: Steht das Archivierungsbit immer noch auf Null, heißt dies, daß seitdem kein Schreibzugriff erfolgte und somit auch keine Veränderung vorgenommen wurde. Die Datei kann somit bei der Sicherung ignoriert werden. Falls das Bit aber zwischenzeitlich gesetzt wurde, kann eine Veränderung stattgefunden haben, so daß die Datei erneut abgearbeitet und das Archivierungsbit auf Null gesetzt werden muß.

Im Datums- und Zeitfeld wird der Zeitpunkt der letzten Änderung (Schreibzugriff) der Datei festgehalten. Das Datum ist dabei wie folgt abgelegt:

Bits 0–4: Tag (1–31)
Bits 5–8: Monat (1–12)
Bits 9–15: Jahr (relativ zu 1980)

Da für das Jahr 7 Bit zur Verfügung stehen, kann DOS Dateien bis ins Jahr $1980+2^7-1=2107$ verwalten. Die Zeit ist im folgenden Format gespeichert:

Bits 0–4: Sekunde (in Zweierschritten)
Bits 5–10: Minute (0–59)
Bits 11–15: Stunde (0–23)

Das Cluster-Feld enthält – wie schon oben beschrieben – die Nummer des Clusters, der die ersten Daten der Datei enthält. Der zugehörige FAT-Eintrag gibt den Folge-Cluster an usw. Sicherlich haben Sie sich gefragt, wie DOS vorgeht, wenn eine Datei erweitert bzw. eine neue Datei angelegt wird. Nun, die FAT wird in der DOS-Version 2 von vorne nach freien Clustern durchsucht, die dann reserviert werden. (In der Version 3 wird übrigens ein anderes Verfahren verwendet, das den Zweck hat, neue Cluster möglichst in der Nähe von schon benutzten Clustern zu reservieren.) Bei einer Dateierweiterung wird dann in den FAT-Eintrag des bisher letzten Clusters die Nummer des ersten neu reservierten Clusters eingetragen, die Endemarkierung erhält den Eintrag des letzten neu reservierten Clusters. Beim Löschen einer Datei werden die belegten Cluster wieder freigegeben, indem die zugehörigen FAT-Einträge mit Nullen gefüllt werden.

Als letzten theoretischen Gesichtspunkt möchte ich nun darauf zu sprechen kommen, wie man aus der Cluster-Nummer die des zugehörigen physikalischen Sektors berechnen kann. Nun, zunächst muß der Anfang des Datenbereichs bestimmt werden: Bei den CCMs existiert ein Boot-Sektor und je nach Kartengröße ein, zwei oder drei Sektoren für die FAT. Jeweils acht weitere Sektoren werden für das Hauptverzeichnis belegt, so daß insgesamt zwölf (32 Kbyte), elf (64 Kbyte) oder zehn (128 Kbyte) Sektoren von dem eigentlichen Datenbereich abgezogen werden müssen. Damit entspricht dem Beginn des Clusters 2 (Cluster 0 und 1 sind ja reserviert) der Sektor 12, 11 bzw. 10. Da jedem Cluster nur ein Sektor zugeordnet ist, kann man die folgenden Sektoren einfach berechnen, indem man sie zu diesen Basiszahlen addiert und 2 abzieht; so hat man

z.B. für den Cluster 5 bei einer 32-Kbyte-Karte die Rechnung 12+5–2 = 15 auszuführen.

Als Beispiel für diese ganze Theorie möchte ich Ihnen nun an Hand einer 32-Kbyte-Karte den Aufbau erläutern. Dafür wurde ein Unterverzeichnis mit dem Namen SUBDIR angelegt, das eine Datei BACKUP.EXE enthält. Neben dem Unterverzeichnis enthält das Hauptverzeichnis noch eine weitere Datei PORTCAPT.EXE. Sehen wir uns zunächst das Hauptverzeichnis an, das ab Sektor 4 zu finden ist:

```
0000  54 45 53 54 44 49 53 4B-20 20 20 28 00 00 00 00   TESTDISK
0010  00 00 00 00 00 00 1A 85-25 00 00 00 00 00 00 00
0020  53 55 42 44 49 52 20 20-20 20 20 10 00 00 00 00   SUBDIR
0030  00 00 00 00 00 00 24 85-25 00 02 00 00 00 00 00
0040  50 4F 52 54 43 41 50 54-45 58 45 20 00 00 00 00   PORTCAPTEXE
0050  00 00 00 00 00 00 68 9A-FB 14 03 00 60 1B 00 00
0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

Als ersten Eintrag findet man die *Volume*-Bezeichnung. Nach dem Namen (11 Zeichen) folgt das Attribut-Byte, wo das Archivierungsbit (5) und das *Volume*-Bit (3) gesetzt sind ($2^5+2^3=32+8=40=\$28$). Als Uhrzeit werden die Bytes $1A85 angegeben. Da zuerst das Low- und dann das High-Byte abgespeichert wird, muß man den Wert in $851A umdrehen. In ein Bitmuster übersetzt sieht es dann so aus:

10000 101000 11010

Damit erhält man die Stunde (10000 = 16), die Minute (101000 = 40) sowie die Sekunde (11010*2 = 28*2 = 56). Die Uhrzeit bei der Formatierung war also 16:40:56. Als nächstes folgt das Datum. Der Wert $2500 muß wieder in $0025 umgedreht werden und sieht im Bitmuster wie folgt aus:

0000000 0001 00101

Man erhält als Jahr 1980 (%0000000+1980), als Monat Januar (0001) und als Tag den 5. (00101), so daß der Formatierungstag mit dem 5.1.1980 angegeben werden kann. Die weiteren Bytes in dem ersten Eintrag sind Null, da sie nicht benötigt werden.

Als nächsten Eintrag haben wir ein Unterverzeichnis mit dem Namen SUBDIR. Man erkennt dies daran, daß Bit 4 des Attribut-Byte gesetzt ist ($10). Die Darstellung des Datums und der Uhrzeit ist jetzt klar, so daß das Augenmerk dem Cluster-Byte gilt: Der Wert 2 besagt, daß das Unterverzeichnis unmittelbar an das Hauptverzeichnis anschließt, auf der 32-Kbyte-Karte also ab dem Sektor 12 folgt.

Als letzten Eintrag haben wir eine Datei PORTCAPT.EXE (der Punkt wird nicht mit abgespeichert), die ab Cluster 3 (Sektor 13 auf der 32-Kbyte-Karte) zu finden ist und deren Dateigröße sich wie folgt berechnet: Das niederwertige Wort enthält die Bytes $60 und $1B, wobei das Low-Byte an erster Stelle kommt. Das höherwertige Wort folgt darauf (Wert: Null). Damit ergibt sich die Gesamtlänge der Datei zu $1B*256+$60 = 7008 Byte.

Um das Unterverzeichnis zu untersuchen, sehen wir uns Sektor 12 an:

```
6000:0000  2E 20 20 20 20 20 20 20-20 20 20 10 00 00 00 00   .
6000:0010  00 00 00 00 00 00 24 85-25 00 02 00 00 00 00 00
6000:0020  2E 2E 20 20 20 20 20 20-20 20 20 10 00 00 00 00   ..
6000:0030  00 00 00 00 00 00 24 85-25 00 00 00 00 00 00 00
6000:0040  42 41 43 4B 55 50 20 20-45 58 45 20 00 00 00 00   BACKUP  EXE
6000:0050  00 00 00 00 00 00 87 9C-FA 14 3A 00 E0 1E 00 00
6000:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

Als ersten Eintrag erkennt man die Datei ».«. An dem Cluster-Byte erkennt man, daß es tatsächlich auf das Unterverzeichnis selbst zeigt, denn auch im Hauptverzeichnis gab es eine 2 zu lesen (s.o.). Die Datei »..« hingegen enthält im Cluster-Byte eine Null, woraus man schließen kann, daß das übergeordnete Verzeichnis (auch Vaterverzeichnis genannt) das Hauptverzeichnis ist. Schließlich existiert noch eine Datei BACKUP.EXE, deren erster Cluster die Nummer $3A ist und die $1E*256+$E0=7904 Byte lang ist. Das Null-Byte am Anfang des nächsten Eintrags kennzeichnet das Ende des Unterverzeichnisses.

Interessant ist jetzt nur noch die FAT:

```
0000  FF FF FF FF 4F 00 05 60-00 07 80 00 09 A0 00 0B
0010  C0 00 0D E0 00 0F 00 01-11 20 01 13 40 01 15 60
0020  01 17 80 01 19 A0 01 1B-C0 01 1D E0 01 1F 00 02
0030  21 20 02 23 40 02 25 60-02 27 80 02 29 A0 02 2B
0040  C0 02 2D E0 02 2F 00 03-31 20 03 33 40 03 35 60
0050  03 37 80 03 39 F0 FF 3B-C0 03 3D E0 03 3F 00 04
0060  41 20 04 43 40 04 45 60-04 47 80 04 49 A0 04 4B
0070  C0 04 4D E0 04 4F 00 05-51 20 05 53 40 05 55 60
0080  05 57 80 05 59 A0 05 5B-C0 05 5D E0 05 5F 00 06
0090  61 20 06 63 40 06 65 60-06 67 80 06 69 A0 06 6B
00A0  C0 06 6D E0 06 6F 00 07-71 20 07 73 40 07 75 60
00B0  07 77 F0 FF 00 00 00 00-00 00 00 00 00 00 00 00
00C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

Das erste $FF-Byte stellt den Media-Descriptor dar, die beiden folgenden Bytes sind Füllbytes. Bevor wir mit der Untersuchung der Einträge beginnen, möchte ich Ihnen noch die Bit-Folge innerhalb der jeweils 12 Bit mitteilen: Man muß hier zwei Fälle unterscheiden. Im ersten Fall beginnt der Eintrag mit einem ganzen Byte und endet mit 4 Bit aus dem nächsten Byte. Hierbei stellen die unteren 4 Bit des zweiten Byte die obersten 4 Bit des Gesamteintrags dar. Im zweiten Fall beginnt der Eintrag mit den obersten 4 Bit des zweiten Byte und endet mit dem kompletten dritten Byte. Hierbei sind die 4 Bit aus dem zweiten Byte die unteren 4 Bit des Gesamteintrags. Kompliziert? Ein wenig schon, deshalb folgendes Beispiel:

05 60 00

Diese 3 Byte sollen an einer durch 3 teilbaren Adresse innerhalb der FAT beginnen, so daß der erste Eintrag mit dem ersten Byte beginnt. Diese 8 Bit stellen die untersten 8 Bit des 12-Bit-Eintrags dar, die obersten noch fehlenden 4 Bit werden aus den unteren 4 Bit des zweiten Bytes gebildet: Da auch hier eine Null steht, ergibt sich für die Gesamtzahl der Wert 005. Der zweite Eintrag dieser 3 Byte holt sich die oberen 8 Bit aus dem dritten Byte. Die unteren 4 Bit gewinnt man aus den oberen 4 Bit des zweiten Byte, so daß sich hier ein Wert von 006 ergibt. Damit ist es einfach, die obige FAT auszuwerten.

Das Unterverzeichnis SUBDIR beginnt im Cluster 2: Hier steht der Wert $FFF (ergibt sich aus dem vierten Byte und den unteren 4 Bit des fünften Bytes der FAT), was nichts anderes bedeutet, als daß kein weiterer Cluster folgt. Die Datei BACKUP.EXE beginnt mit dem Cluster 3: Hier kann man nun die Clusterkette weiterverfolgen. Es werden die Cluster $004, $005, $006, $007, ..., $037, $038 und $039 belegt. Im Cluster $040 selbst (Adresse $0055 die oberen 4 Bit, Adresse $0056) steht der Wert $FFF, es folgen also keine weiteren Cluster mehr. Da oben gesagt wurde, daß DOS der Version 2 immer den nächsten freien Cluster belegt, verwundert diese Belegung nicht.

Schließlich folgt noch die Belegung der Datei PORTCAPT.EXE, die sich ab Cluster $03A direkt anschließt und bis zum Cluster $077 reicht. Ab der Adresse $00B2 folgt wieder der Abschluß mit dem Wert $FFF. Sie sehen, daß eine Analyse der CCMs gar nicht so schwierig ist, wenn man den Aufbau kennt. Die einzige noch offengebliebene Frage ist die, wie man vom Programm aus direkt auf einzelne Sektoren zugreifen kann. Natürlich könnte man direkt den Speicher ansprechen, indem man die Position des Sektors berechnet: Diese Programme sind aber auf dem PC nicht austestbar, so daß man sich der BIOS-Interrupts bedienen sollte.

Um ein CCM zu formatieren, kann die Funktion $07 des Interrupt $61 benutzt werden. Dazu wird einfach im AL-Register die Drive-Nummer übergeben (0=A:, 1 = B:). Falls ein Fehler auftreten sollte, wird das Carry-Flag gesetzt und im AH-Register der Fehlercode zurückgeliefert (s.u.). Diese Funktion sollte nicht dazu benutzt werden, die RAM-Disk C: zu formatieren. Hierfür hat DIP eine eigene Funktion ($09) eingerichtet. Im BX-Register wird die gewünschte Größe in Kbyte übergeben. Falls erfolgreich formatiert werden konnte, wird das System neu gebootet, wie Sie es auch nach dem FDISK-Befehl gewohnt sind. Im Fehlerfall (gesetztes Carry-Flag, in BX wurde ein zu großer Wert übergeben) liefert das BX-Register die maximal mögliche Größe in Kbyte zurück.

Alle Lese-Schreib-Operationen bei CCMs können einen Fehlercode erzeugen, der durch ein gesetztes Carry-Flag angezeigt und stets im Register AH zurückgeliefert wird. Dabei sind die folgenden Codes möglich:

$01: Nicht erlaubte Funktionsnummer übergeben
$03: CCM schreibgeschützt (nur bei Schreibzugriffen)
$04: Sektor nicht gefunden (nur Nummern 1 bis 8 erlaubt!)
$09: Datenübertragung über Segmentgrenze
$10: Lesefehler
$40: Spur nicht gefunden
$80: Keine CCM vorhanden

Beim PC existieren noch einige weitere Fehlercodes, die sich auf die Hardware beziehen (DMA-Controller, Disk-Controller), welche beim Portfolio nicht vorhanden ist und damit auch keinen Fehler erzeugen kann.

Der Fehlercode $03 tritt durch einen BIOS-Fehler bei direkten Sektorzugriffen über BIOS-Interrupts *nicht* auf, auch wenn das CCM schreibgeschützt ist. Sie sollten sich daher auf keinen Fall darauf verlassen, daß die Daten eines schreibgeschützten CCM auch wirklich geschützt sind – dies ist nur bei Schreibzugriffen über DOS der Fall.

Im folgenden möchte ich die Interrupt-Funktionen beschreiben, die auf eine einzelne Spur zugreifen. Leider kann man diesen Funktionen nicht einfach die Tracknummer (0 bis 31) übergeben, sondern muß sie in Seite und Zylinder aufteilen. Da die ersten 16 Tracks auf der Oberseite liegen, die 16 unteren auf der Unterseite (was für ein Unsinn bei einer RAM-Karte!), kann man Seite und Zylinder nach diesen Formeln berechnen:

Seite = INT(Track/16)

Zylinder = Track – 16xSeite

Für alle folgenden Funktionen müssen diese Parameter an die Interrupt-Service-Routine übergeben werden: Im Register DL die Drive-Nummer (0=A:, 1=B:, 2=C:), in DH die Seite und in CH die Zylindernummer. Um einen Track eines CCM zu formatieren, kann die Funktion $05 des BIOS-Interrupt $13 benutzt werden. Eine Ausnahme bildet der Track 0: Da die Funktion $05 keinen Boot-Sektor anlegt, muß hierfür die Funktion $83 benutzt werden. Im Gegensatz zum PC können nicht einzelne Sektoren, sondern immer nur ganze Tracks formatiert werden. Das Füll-Byte kann dabei der Disk-Table (Interrupt $1E, Offset $0A) entnommen werden, im Normalfall steht hier $F6. Ebenfalls in der Disk-Table (Offset $03) ist ein Code für die Sektorgröße eingetragen:

0: 128 Byte pro Sektor
1: 256 Byte pro Sektor
2: 512 Byte pro Sektor

Die nächsten drei Funktionen des Interrupt $13 dienen dem Lesen ($02), Beschreiben ($03) und Verifizieren ($04) eines oder mehrerer Sektoren. Dazu müssen zusätzlich zu den oben aufgeführten Parametern noch vier weitere übergeben werden: Im Register CL die Nummer des ersten zu bearbeitenden Sektors, in AL die Anzahl der zu bearbeitenden Sektoren sowie in ES und BX die Segment- und Offsetadresse des Datenpuffers, in den die Daten eingelesen bzw. aus dem sie auf das CCM geschrieben werden. Wichtig ist zu wissen, daß immer nur aufeinanderfolgende Sektoren innerhalb eines Tracks bearbeitet werden können:

CL=1, AL=5:
Korrekt, die Sektoren 1 bis 5 eines Tracks werden bearbeitet.
CL=5, AL=4:
Inkorrekt, da auf einem Track nur Sektornummern bis 8 zulässig sind (5+4 = 9!).

Wichtig ist ebenfalls, daß während des Datentransfers kein Segmentüberlauf eintritt, da sonst mit dem Fehlercode $09 abgebrochen wird:

ES=$4000, BX= $1000, CL=1, AL = 8:
Korrekt, die Daten (8x512=4096 Bytes) werden von $4000:$1000 bis $4000:$0FFF abgelegt.
ES=$4000, BX= $FF00, CL=1, AL = 8:
Inkorrekt, die Daten werden bei $4000:$FFFF über die Segmentgrenze geschrieben.

Über die Funktion $00 kann ein Reset des CCM ausgelöst werden, wobei im DL-Register die Drive-Nummer übergeben wird. Während dieser

Funktionsaufruf beim PC nach einem Fehler unmittelbar nötig ist, kann man beim Portfolio darauf verzichten. Über die Funktion $01 schließlich kann man den Status holen, man liest also den Fehlercode aus, soweit er vorhanden ist, ohne eine bestimmte Aktion durchzuführen. Auch hier muß in DL die Drive-Nummer übergeben werden.

Zum Schluß noch ein paar Beispiele:

AH = $00, DL = $00, Aufruf INT $13:
Reset von CCM-Laufwerk A:

AH = $01, DL = $02, Aufruf INT $13:
Status der RAM-Disk C: holen.

AH=$02, DL=$00, DH=$01, CH=$01, CL=$01, AL=$04, ES=$2000, BX=$0000, Aufruf INT $13:
Die Sektoren 1 bis 4 des Tracks 17 des CCM A: werden in den Buffer ab $2000:$0000 eingelesen.

AH=$03, DL=$01, DH=$00, CH=$0B, CL=$03, AL=$01, ES=$2000, BX=$0000, Aufruf INT $13:
Der Sektor 3 des Tracks 11 des CCM B: wird mit dem Bufferinhalt ab $2000:$0000 beschrieben.

AH=$04, DL=$02, DH=$00, CH=$04, CL=$01, AL=$08, ES=$2000, BX=$0000, Aufruf INT $13:
Die Sektoren 1 bis 8 des Tracks 4 der RAM-Disk C: werden mit dem Bufferinhalt ab $2000:$0000 verglichen.

AH=$05, DL=$00, DH=$01, CH=$0F, Aufruf INT $13: Track 31 des CCM A: wird formatiert.
AH=$83, DL=$00, DH=$00, CH=$00, Aufruf INT $13: Track 0 des CCM A: wird formatiert und ein Boot-Sektor angelegt.

1.5 Die Tastatur

Nach dem Bildschirm stellt die Tastatur die zweite große Inkompatibilitätsquelle des Portfolio gegenüber dem PC dar. Viele Programme gehen davon aus, daß sich an der Adresse $60 der Port für den Tastatur-Controller befindet und programmieren diesen auf direkte Weise. Beim Portfolio suchen sie aber vergebens, so daß diese Programme nicht funktio-

nieren können. Aber auch andere Unterschiede müssen beachtet werden; dazu später.

Zunächst möchte ich Ihnen nahebringen, was eigentlich passiert, wenn Sie eine Taste betätigen: Zunächst bekommt der Prozessor davon gar nichts mit, nur der Tastatur-Controller weiß Bescheid. Er ist jedoch in der Lage, einen Hardware-Interrupt auszulösen, den Interrupt $09. Die Service-Routine dieses Interrupts ist nun dafür einzig und allein verantwortlich, daß der Tastendruck ausgewertet wird. Dafür wird der Tastatur-Controller angesprochen und ein entsprechender Zeichencode ausgelesen. Dieser hat beim Portfolio zunächst überhaupt nichts mit dem bekannten ASCII- oder Scan-Code zu tun, sondern wird erst softwaremäßig umgewandelt.

Zunächst prüft die Routine, ob ein sogenannter Hot-Key gedrückt wurde, also eine Taste bzw. eine Tastenkombination, die bestimmte Aktionen starten soll. Beim Portfolio sind dies z.B. die Tasten [W] oder [E] in Verbindung mit der [Atari]-Taste (Aufruf der Applikationen) oder die Kombination [Fn]+[P] (Hardcopy), wobei im letzteren Fall der Interrupt 5 aufgerufen wird. Durch [Fn]+[B] (beim PC: [Strg]+[Break]) wird der Interrupt $1B aufgerufen, wodurch im Normalfall das laufende Programm unterbrochen und zurück ins DOS verzweigt wird. Im Gegensatz zu den übrigen Tasten werden diese Codes intern nicht gespeichert, so daß man sie nicht von einem Programm abfragen kann.

Falls die Prüfung ergeben hat, daß kein Hot-Key betätigt wurde, werden Scan- und ASCII-Code in den Tastaturpuffer eingetragen, soweit es sich nicht um einen erweiterten Tastaturcode handelt. Einige Zeichen wie z.B. die Cursor-Tasten haben in dem 256 Zeichen umfassenden ASCII-Zeichensatz nämlich keinen Platz mehr gefunden, so daß sie gesondert behandelt werden müssen. Hierbei wird dann statt des ASCII-Codes eine Null abgespeichert, der Scan-Code wird durch den erweiterten Tastaturcode ersetzt. Bei dem Tastaturpuffer handelt es sich um einen 16 Zeichen fassenden Ringpuffer, der 32 Byte groß ist und von $0040:$001E bis $0040:$003D liegt. Wie bei einem Ringpuffer üblich können ständig neue Zeichen eingefügt bzw. alte ausgelesen werden. Die Adresse des nächsten auszulesenden Zeichens steht dabei in dem Speicherwort an der Adresse $0040:$001A. Wird ein Zeichen ausgelesen, wird der Zeiger so lange um 2 Byte erhöht, bis das Ende des Puffers erreicht ist, dann wird er wieder auf den Anfang des Puffers gesetzt. Der nächste freie Platz im Puffer wird durch das Wort in der Adresse $0040:$001C angegeben. Wird ein neues Zeichen eingegeben, wird es an der Stelle abgelegt, die durch diesen Zeiger

festgelegt wird. Anschließend wird auch er um 2 Byte erhöht, bis das Ende des Tastaturpuffers erreicht ist. Dann wird er wieder auf den Anfang zurückgesetzt (Bild 1.11).

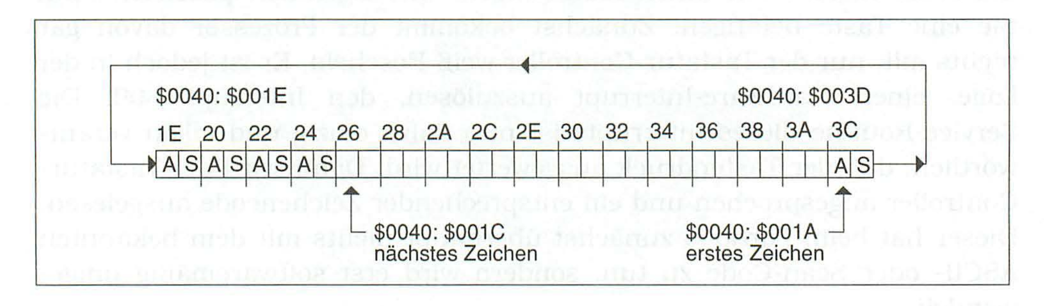


Bild 1.11: Aufbau und Funktion des Tastaturpuffers

Zwei Fälle verdienen besondere Beachtung: Zum einen kann der Fall auftreten, daß beide Zeiger auf die gleiche Adresse zeigen. In diesem Fall ist der Tastaturpuffer leer. Zum anderen kann der Zeiger auf die nächste freie Adresse ($001C) genau 2 Byte vor dem des nächsten auszulesenden Zeichens stehen ($001A). Wenn nun ein Zeichen eingelesen wird, wird er auf dessen Adresse erhöht. In diesem Fall ist der Tastaturpuffer voll, da das nächste eingegebene Zeichen ja das erste auszulesende überschreiben würde. Damit dies verhindert wird, wird im Moment der Bewegung des Zeigers von $001C auf den Wert von $001A ein Flag gesetzt, das die Meldung »Puffer voll« repräsentiert. Die Interrupt-Service-Routine reagiert daruf mit dem bekannten Piepston, um zu signalisieren, daß der Tastaturpuffer gefüllt ist. Sowie ein Zeichen ausgelesen wird, wird das Flag wieder gelöscht, so daß weitere Zeichen aufgenommen werden können.

Bei den »normalen« ASCII-Codes muß man zwischen den »wirklich normalen« und den Steuercodes unterscheiden. Durch diese werden bestimmte Aktionen veranlaßt:

Steuercodes des Portfolio	
8	Backspace
9	Tabulator
10	Linefeed (Zeilenvorschub)
13	Return (Cursor an Zeilenanfang)
27	Escape

Bild 1.12: Steuercodes im ASCII-Zeichensatz

Die Scan-Codes bzw. die erweiterten Tastaturcodes der Portfolio-Tastatur zeigt Bild 1.13. Bei mehrfach belegten Tasten ist in der oberen Reihe immer der Code für die Sonderbelegung aufgeführt (z.B. bei der Taste [←] oben der Code für [Pos1], unten für [←]).

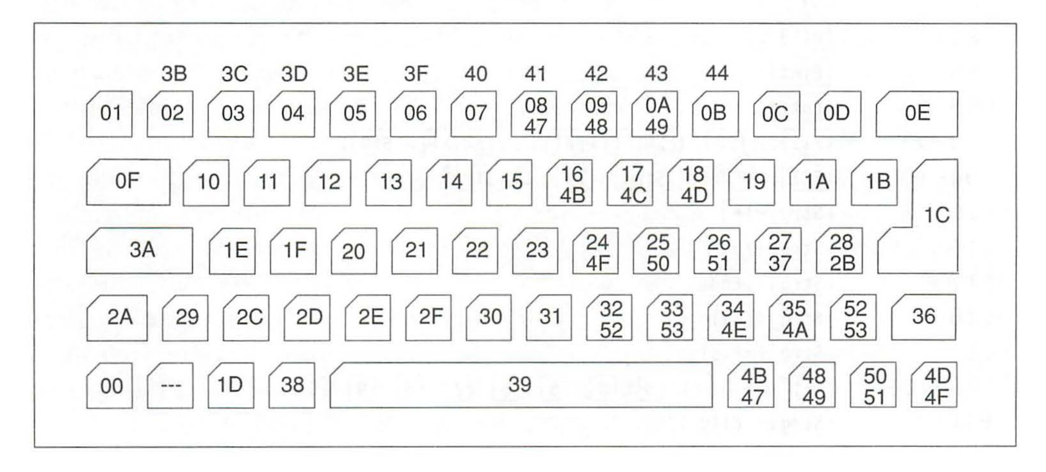


Bild 1.13: Scan-Codes der Portfolio-Tastatur

Beim PC ist es möglich, durch die Kombination der [Alt]-Taste mit den Ziffern des Zehnerblocks jeden beliebigen ASCII-Code direkt einzugeben. Beim Portfolio kann man zwar diese Methode ebenfalls anwenden, indem man den Zehnerblock simuliert, leider versagt sie aber bei den Codes von 1 bis 31, also gerade bei denen, die man nicht direkt eingeben kann und für die diese Methode ursprünglich gedacht war! Das ist schon bitter, aber nicht zu ändern. Dafür funktioniert das Verfahren wenigstens bei den übrigen Codes, die größer als 31 sind.

Ebenfalls korrekt implementiert sind die erweiterten Tastaturcodes:

Erweiterte Tastaturcodes des Portfolio	
16-25	Alt+Q W E R T Y U I O P
30-38	Alt+A S D F G H J K L
44-50	Alt+Z X C V B N M
59-68	F1...F10
71	Pos1
72	↑
73	Bild ↑
75	←
77	→
79	Ende
80	↓
81	Bild ↓
82	Einf
83	Entf
84-93	F11..F20 (SHIFT+F1...SHIFT+F10)
94-103	F21..F30 (Strg+F1...Strg+F10)
115	Strg+←
116	Strg+→
117	Strg+Ende
118	Strg+Bild ↓
119	Strg+Pos1
120-129	Alt+1 2 3 4 5 6 7 8 9 0
131	Strg+Bild ↑

Bild 1.14: Die erweiterten Tastaturcodes des Portfolio

Für die Kommunikation mit der Tastatur stellt das BIOS drei Funktionen zur Verfügung, die allesamt über den Interrupt $16 aufgerufen werden. Die Funktion $00 liest ein Zeichen aus dem Tasturpuffer aus und entfernt es aus diesem. Falls sich kein Zeichen im Puffer befindet, wird so lange gewartet, bis ein Zeichen eingegeben wird. Die Rückgabe erfolgt im AX-Register: Handelt es sich um einen erweiterten Tastaturcode, wird dieser im Register AH übergeben, während das AL-Register den Wert 0 aufweist. Ist hingegen eine normale Taste betätigt worden, wird im AL-Register der ASCII-Code und im AH-Register der Scan-Code der jeweiligen Taste zurückgeliefert.

Die Funktion $01 hingegen prüft, ob ein Zeichen im Tastaturpuffer vorhanden ist: Falls dies der Fall ist, wird das Zero-Flag gelöscht und der Code wie bei der Funktion $00 im AX-Register zurückgeliefert. Im Unterschied dazu wird es jedoch nicht aus dem Tastaturpuffer entfernt, so daß man bei einem erneuten Aufruf genau das gleiche Zeichen ausliest. Falls also ein Zeichen vorhanden ist, muß man anschließend die Funktion $00 aufrufen, um es auch wirklich zu entfernen. Sollte kein Zeichen im Tastaturpuffer vorhanden sein, wird das Zero-Flag gesetzt. Mit dieser Funktion kann man die Tastatur »pollen«, d.h., ständig abfragen, ob ein Zeichen eingegeben wurde. Im Gegensatz zur Funktion $00 behält dabei das eigene Programm die Kontrolle, so daß zwischen zwei Abfragen noch andere Aufgaben erledigt werden können. Der Kommandoprozessor des PC benutzt z.B. die Funktion $01, um zwischendurch eine Hintergrund-Druckausgabe durch den PRINT-Befehl durchführen zu können.

Der Kommandoprozessor des Portfolio benutzt jedoch die Funktion $00. Der Grund ist einfach zu erklären: Die Interrupt-Funktion $00 besitzt für den Portfolio eine viel weitreichendere Bedeutung als für den PC: Mit ihrem Aufruf wird nämlich ein Timer gestartet, nach dessen Ablauf die sogenannte Power-down-Sequenz steht, an deren Ende für jedermann sichtbar das Abschalten des Bildschirms steht. Dadurch wird ein enormer Stromspar-Effekt erzielt. Das DOS verzichtet damit aber auch auf jede Möglichkeit der Hintergrundsteuerung, da die Kontrolle während des Wartens auf einen Tastendruck ausschließlich beim BIOS liegt. Die bitterste Konsequenz für Programmierer besteht aber darin, daß mit der Power-down-Sequenz auch gewisse Hardware-Abschnitte des Portfolio quasi lahmgelegt werden. So ist es z.B. nicht mehr möglich, externe Interrupts über die serielle Schnittstelle zu empfangen. Wenn man also auf diese Hardware angewiesen ist, muß man die Funktion $01 benutzen, wodurch die Einleitung der Power-down-Sequenz verhindert wird. Die eigentliche Tragik liegt nun darin, daß alle Einlesefunktionen von Hochsprachen (z.B. *READLN* von Turbo Pascal) im Endeffekt die BIOS-Funktion $00 benutzen und deshalb in diesen speziellen Fällen unbrauchbar sind. Der Programmierer muß sich dann eine eigene Eingaberoutine schreiben, die die Funktion $01 benutzt. Dies ist äußerst ärgerlich, aber nicht zu vermeiden. Ebenso wird die TSR-Programmierung durch die Nutzung der Funktion $00 durch den Kommandoprozessor wesentlich erschwert, wie in Kapitel 2 gezeigt wird. Aber DIP hat sich beim Portfolio nun mal das Ziel »Stromsparen« als höchste Priorität auserkoren, und dafür ist die Power-down-Sequenz eine wirklich geniale Erfindung!

Durch die Funktion $02 schließlich kann der Tastaturstatus erfragt werden, der im Register AL zurückgeliefert wird. Dabei haben die 8 Bit folgende Bedeutung:

Bit 0 gesetzt: Rechte Shift-Taste betätigt
Bit 1 gesetzt: Linke Shift-Taste betätigt
Bit 2 gesetzt: Strg-Taste betätigt
Bit 3 gesetzt: Alt-Taste betätigt
Bit 4 gesetzt: Scroll-Lock an
Bit 5 gesetzt: Num-Lock an
Bit 6 gesetzt: Shift-Lock an
Bit 7 gesetzt: Entf-Lock an

Den Status kann man auch holen, indem man die BIOS-Variable an der Adresse $0040:$0017 ausliest. Die Interrupt-Funktion $02 macht auch nichts anderes, als gerade dieses Byte zu lesen.

Ein immerwiederkehrendes Problem bei der Softwareentwicklung sind zu lange Eingaben der Benutzer. Stellen Sie sich vor, das Programm fordert Sie auf, entweder die Taste [1] oder [2] zu drücken, Sie betätigen aber beide Tasten. Nun, zunächst wird Ihre Auswertungsroutine nur das erste Zeichen holen und auswerten, wodurch die erstgedrückte Taste, also die [1], zur Wirkung käme. Was aber passiert, falls das Programm eine weitere Eingabe der Form »[1] oder [2]« fordert? Da die [2] noch als Restmüll im Tastaturpuffer verbleibt, wartet das Programm gar nicht ab, bis Sie eine Eingabe getätigt haben, sondern liest die verbliebende [2] aus! Um diese unangenehme Begleiterscheinung zu verhindern, kann man zwei Dinge tun: Zum einen kann man nach jeder Eingabe die Pointer des Tastaturpuffers auf die gleiche Adresse, z.B. den Anfang des Puffers setzen, um dem System vorzugaukeln, der Puffer sei leer (das Beschreiben des Tastaturpuffers vom Programm aus ist möglich und ermöglicht spektakuläre Programme; siehe Abschnitt 2.9 »Die Nachlade-Unit«), zum anderen kann man eine Funktion des DOS-Interrupt $21 benutzen ($0C), die zunächst den Tastaturpuffer leert und dann eine DOS-Eingabefunktion nach Wahl aufruft: Es dürfen die DOS-Funktionen $01, $06, $07, $08 und $0A aufgerufen werden, wobei die Funktionsnummer der Funktion $0C im AL-Register übergeben wird. Bis auf die Funktion $0A (Eingabe einer Zeichenkette in einen Puffer) wird das eingegebene Zeichen im AL-Register zurückgeliefert. Die DOS-Eingabefunktionen arbeiten allesamt mit dem BIOS-Interrupt $16, Funktion $00, so daß auch hier bei längerem Warten auf einen Tastendruck die Power-down-Sequenz eingeleitet

wird. Falls man auf das Polling angewiesen ist, muß man daher den Puffer von Hand löschen:

$0040:$001A = $001E und $0040:$001C = $001E

Die erwähnten DOS-Interrupt-Funktionen sind im Anhang beschrieben.

1.6 Die serielle Schnittstelle

Die Bedeutung der seriellen Schnittstelle ist beim Portfolio ungleich größer als beim PC, da sie neben der normalen Funktion als »Datenüberträger« als einziges serienmäßiges Zubehör die Möglichkeit bietet, externe Interrupts an den Prozessor weiterzuleiten; dazu aber später.

Zunächst möchte ich auf die normale Datenübertragung zu sprechen kommen. Wie Sie sicherlich wissen, stellt die serielle Übertragung die universellste Möglichkeit überhaupt dar: Nicht nur mit Druckern oder anderen Computern kann kommuniziert werden, sondern über Modem bzw. Akustikkoppler kann per Telefon praktisch jeder Winkel dieser Erde erreicht werden, indem die elektrischen Signale in akustische umgewandelt werden. Der einzige Nachteil ist, wie der Name »seriell« schon sagt, daß die einzelnen Bits nicht parallel, sondern hintereinander übertragen werden und ein relativ kompliziertes Übertragungsprotokoll eingehalten werden muß, so daß eine serielle Übertragung insgesamt ziemlich langsam abläuft. Zumindest bei Nadel-Druckern ist dies aber ohne Bedeutung, da diese wesentlich langsamer ausdrucken als Daten übertragen werden können. Ein weiterer Vorteil besteht darin, daß nur sehr wenige Leitungen vorhanden sein müssen, also Kosten gespart werden.

Die Daten werden dabei über eine einzige Leitung übertragen, die einen der beiden Zustände »logisch 0« oder »logisch 1« annehmen kann. Ob dabei »logisch 1« bedeutet, daß eine hohe Spannung anliegt oder nicht (»physikalisch 1« bzw. »physikalisch 0«), spielt keine Rolle. Im Normalfall steht die Leitung auf »Logisch 1«. Soll nun ein Zeichen übertragen werden, zieht die Hardware (ein UART genannter Prozessor) die Leitung auf »Logisch 0«, worauf der Empfänger weiß, daß ein Zeichen übertragen werden soll. Hierbei handelt es sich um das Startbit der Übertragung. Die Länge jedes Bits hängt von der Übertragungsgeschwindigkeit ab, die in Baud (Bit pro Sekunde) gemessen wird. Je mehr Baud man wählt, desto

schneller kann übertragen werden, desto größer ist aber auch die Fehlerwahrscheinlichkeit.

Nach dem Startbit werden dann die Datenbits übertragen, wobei mit dem niederwertigsten Bit begonnen wird. Man kann zwischen 7 oder 8 Datenbit auswählen. Da man mit 7 Bit nur ASCII-Codes von 0 bis 127 darstellen kann, benötigt man im Deutschen wegen der Umlaute, deren ASCII-Code größer als 128 ist, immer 8 Datenbit. Bei einem gesetzten Bit erhält die Leitung den Zustand »Logisch 1«, sonst »Logisch 0«. Nach den Datenbits folgt optional das Parity-Bit. Hierbei wird die Anzahl der gesetzten Datenbits durch das Parity-Bit so ergänzt, daß insgesamt eine gerade (gerade Parität) oder ungerade (ungerade Parität) Anzahl gesetzter Bits vorhanden ist. Dadurch kann der Empfänger etwaige Übertragungsfehler feststellen: Wurde z.B. eine gerade Parität gewählt, während insgesamt aber 3 gesetzte Bit (Datenbits + Parity-Bit) empfangen wurden, kann der Empfänger von einem Übertragungsfehler ausgehen und dies dem Sender mitteilen. Leider stellt dies Verfahren aber nicht das Optimum dar, denn bei mehreren Fehlern können sich diese kompensieren; wenn z.B. zwei ursprünglich gesetzte Bits beide in nicht gesetzte umgewandelt werden, wird an der Parität nichts verändert.

Nach dem Parity-Bit folgen noch 1 bis 2 Stop-Bit, wobei die Leitung wieder auf »Logisch 1« gesetzt wird. Auch 1,5 Stop-Bit sind möglich, was nichts anderes bedeutet, als daß die Leitung eineinhalbmal solange auf »Logisch 1« bleibt, wie durch die Baud-Rate für ein Bit festgelegt wurde. So bedeuten z.B. 1,5 Bit bei einer Übertragungsrate von 300 Baud (1 Bit = $^{1}/_{300}$ Sekunde) einen Zeitraum von $^{1,5}/_{300}$ = $^{1}/_{200}$ Sekunde. Jetzt ist die Übertragung eines Zeichens abgeschlossen. Bild 1.15 zeigt den Verlauf der Übertragung des Namenszuges DIP bei gerader Parität. (Sie sehen, diese Firma hat es mir angetan; in der Tat ist der Portfolio schon ein kleiner Geniestreich...)

Um die Übertragung zu realisieren, besitzt der UART vier Register: Beim Senden wird das Datenbyte in das Transmission-Holding-Register geschrieben, von wo es in das Transmission-Shift-Register übertragen wird. Dabei werden je nach Wahl Start-, Paritäts- und Stopp-Bits eingefügt. Aus diesem Register werden dann die einzelnen Bits nach und nach auf die Leitung ausgegeben. Der hierfür erforderliche Zeittakt wird durch einen Quarz erzeugt, das mit 1,8432 MHz getaktet ist und dessen Frequenz entsprechend der gewählten Baud-Rate heruntergeteilt wird.

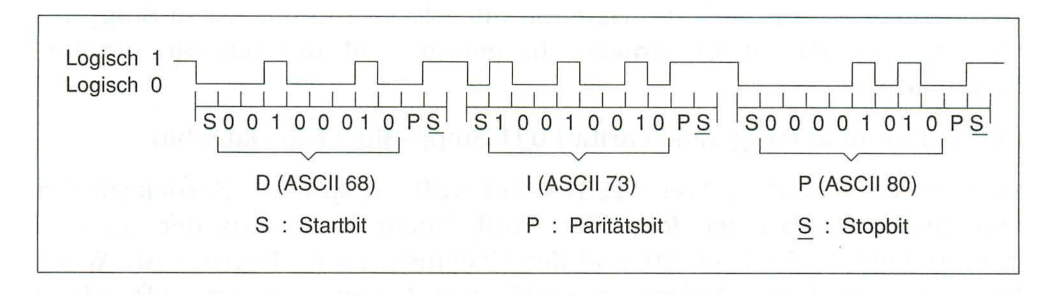


Bild 1.15: Serielle Zeichenübertragung

Beim Empfang von Daten werden diese in das Receiver-Shift-Register geladen, wobei Start-, Parity- und Stop-Bits entfernt werden. Das eigentliche Datenbyte wird nach abgeschlossener Übertragung in das Receiver-Data-Register geladen, von wo aus es der Programmierer abholen kann. Um nicht direkt die Hardware des UART programmieren zu müssen, stellt das BIOS insgesamt vier Funktionen des Interrupt $14 zur Verfügung, mit denen eine serielle Übertragung einfach realisiert werden kann.

Die Funktion $00 übernimmt die Initialisierung der Schnittstelle. Dafür wird im DX-Register die Nummer der Schnittstelle übergeben (im Portfolio immer Null), im Register AL müssen die Konfigurationsparameter stehen. Hierbei existieren vier Funktionsgruppen, die jeweils 1 bis 3 Bit enthalten:

Bit-Belegung des Konfigurationsregisters AL

	Übertragung		Parität		Stoppbits		Datenbits	Gruppe
7 6 5		4 3		2		1 0		Bit-Nr.
0 0 0	110 Baud	0 0	Keine	0	1	0 0	——	
0 0 1	150 Baud	0 1	Gerade	1	Nach Baud-	0 1	——	
0 1 0	300 Baud	1 0	——		Rate 1, 1,5	1 0	7 Bit	
0 1 1	600 Baud	1 1	Ungerade		2 Bits	1 1	8 Bit	
1 0 0	1200 Baud							
1 0 1	2400 Baud							
1 1 0	4800 Baud							
1 1 1	9600 Baud							

Bild 1.16: Belegung des Konfigurationsregisters

Nehmen wir an, Sie wollen 8 Datenbit mit 1200 Baud mit einem Stopp-Bit und gerader Parität übertragen: In diesem Fall müßten Sie die Bitkombination

100 (8 Datenbit) 01 (gerade Parität) 0 (1 Stopp-Bit) 11 (8 Datenbit)

übergeben, was dem Wert 128+8+2+1=139 entspricht. Zurückgeliefert wird (wie auch bei den folgenden Funktionen) der Status der seriellen Schnittstelle im Register AH und der Modemstatus im Register AL. Wenn Sie natürlich kein Modem angeschlossen haben, sondern z.B. einen Drucker, bleiben einige Bits ohne Bedeutung. Zunächst zum Status der seriellen Schnittstelle:

Bit 0 gesetzt: Daten wurden empfangen und in Empfangspuffer geschrieben
Bit 1 gesetzt: Daten im Empfangspuffer wurden überschrieben
Bit 2 gesetzt: Paritäts-Fehler
Bit 3 gesetzt: Zeichen wurde ohne Stopp-Bit empfangen
Bit 4 gesetzt: Für die Dauer einer Zeichenübertragung wurden nur Nullen empfangen
Bit 5 gesetzt: Transmission-Holding-Register leer
Bit 6 gesetzt: Transmission-Shift-Register leer, Zeichen wurde gesendet

Bevor ich auf den Modem-Status zu sprechen komme, lassen Sie mich noch auf zwei Probleme bei der seriellen Übertragung hinweisen, die mit dem Status der Schnittstelle zusammenhängen: Beim Senden von Daten muß man aufpassen, daß man nicht in das Transmission-Holding-Register schreibt, während dort noch das letzte Zeichen steht, da dieses sonst überschrieben würde. Man muß also so lange warten, bis Bit 5 des Status-Bytes gesetzt ist. Viel schlimmer kann es aber beim Empfang von Daten kommen: Wenn man diese nicht rechtzeitig aus dem Receiver-Data-Register abholt, kann es passieren, daß sie vom nächsten empfangenen Zeichen überschrieben werden. Zwar wird einem das durch das gesetzte Bit 1 mitgeteilt, das Datenbyte ist aber unwiederbringlich verlorengegangen. Um so etwas zu vermeiden, hat man eine Möglichkeit vorgesehen, um nach dem Empfang des Datenbytes einen Interrupt auszulösen, so daß das Programm sofort reagieren und das Byte auslesen kann. Hätte man diese Möglichkeit nicht, müßte das Programm ständig die Schnittstelle abfragen (»Polling«), wodurch keine anderen Aufgaben mehr wahrgenommen werden könnten.

Nun aber zum Modemstatus. Leider muß man sich dazu etwas mit der Hardware beschäftigen. Wenn ich es in diesem Buch auch so oft wie möglich vermeide, direkt auf die Hardware zu sprechen zu kommen (dafür gibt

es ja die BIOS- und DOS-Routinen) – in diesem Fall läßt es sich für das Verständnis nicht vermeiden.

Ursprünglich war die serielle Schnittstelle für die Datenübertragung zu einem Modem gedacht. Es verwundert also nicht, wenn die Schnittstelle mit Leitungen belegt ist, die auf die Bedienung eines Modems geradezu zugeschnitten sind. Für die Bedienung eines Druckers z.B. sind diese Leitungen nicht allesamt erforderlich. Hier nun die Belegung des neunpoligen Sub-D-Stecker des Portfolio:

Pin 1: CD (Carrier Detect)
Pin 2: RD (Receive Data)
Pin 3: TD (Transmit Data)
Pin 4: DTR (Data Terminal Ready)
Pin 5: GND (Signal Ground)
Pin 6: DSR (Data Set Ready)
Pin 7: RTS (Request To Send)
Pin 8: CTS (Clear To Send)
Pin 9: RI (Ring Indicator)

Fangen wir mit den einfachen Leitungen an: Während GND die Masseverbindung darstellt, werden die Daten vom Computer über die Leitung TD gesendet und über die Leitung RD empfangen. Die restlichen Leitungen dienen dem »Handshaking« mit dem Modem, ihre Aufgabe besteht darin, eine störungsfreie Kommunikation zu ermöglichen. So zeigt der Rechner dem Modem über die Leitung RTS an, daß es mit dem Senden der Daten beginnen soll. Über die Leitung CTS zeigt das Modem an, daß es frei ist, Daten an die Gegenstation (am anderen Ende der Telefonleitung) zu übertragen. Diese beiden Leitungen werden für den Druckerbetrieb nicht direkt benötigt und daher verbunden: Wird RTS vom Computer auf »logisch 1« gelegt, bekommt er über den Eingang CTS ebenfalls eine »Logisch 1« präsentiert und »weiß« somit, daß der Drucker frei ist. Über die Leitung DSR zeigt das Modem an, daß es betriebsbereit ist, über DTR meldet der Rechner, daß er betriebsbereit ist. Diese Leitung bleibt im Druckerbetrieb unbelegt, da es für den Drucker ja uninteressant ist, ob der Rechner betriebsbereit ist. Umgekehrt wird jedoch die DTR-Leitung des Druckers mit der DSR-Leitung verbunden, um dem Rechner anzuzeigen, daß der Drucker betriebsbereit ist. Die Leitung RI stellt eine Klingelanzeige dar und meldet, wenn das Telefon klingelt, mit dem das Modem verbunden ist. Die Leitung CD schließlich gibt den Empfangspegel der Gegenstation an, was wichtig für die Interpretation der Leitungszustände als »Logisch 0« oder »Logisch 1« ist. Die letzten beiden Leitungen

sind im Druckbetrieb ebenfalls unbelegt. Mit diesem Hintergrundwissen kann man sich nun der Belegung des Modemstatusregisters zuwenden:

Bit 0 gesetzt: CTS-Leitung änderte ihren Zustand seit letztem Zugriff
Bit 1 gesetzt: DSR-Leitung änderte ihren Zustand seit letztem Zugriff
Bit 2 gesetzt: RI-Leitung änderte ihren Zustand seit letztem Zugriff
Bit 3 gesetzt: CD-Leitung änderte ihren Zustand seit letztem Zugriff
Bit 4: Entspricht dem CTS-Signal
Bit 5: Entspricht dem DSR-Signal
Bit 6: Entspricht dem RI-Signal
Bit 7: Entspricht dem CD-Signal

Die ersten 4 Bit geben an, ob sich ein Leitungszustand (Eingang) seit dem letzten Auslesen des Modemstatus geändert hat. Damit kann man bei zwei aufeinanderfolgenden Zugriffen nicht nur feststellen, ob z.B. das Modem sendebereit ist (Bit 4 gesetzt), sondern auch, ob dies schon länger der Fall (Bit 0 gelöscht) oder gerade eben erst zustande gekommen ist (Bit 0 gesetzt).

Mit der Interrupt-Funktion $01 kann ein Zeichen übertragen werden. Dafür wird der ASCII-Code im Register AL übergeben, im DX-Register muß beim Portfolio eine Null stehen (Nummer der seriellen Schnittstelle). Zurückgeliefert wird im AH-Register der Status der Schnittstelle, wobei ein gelöschtes Bit 7 bedeutet, daß das Zeichen übertragen wurde. Ist Bit 7 hingegen gesetzt, trat ein Fehler auf, dessen Ursache in den Bits 0 bis 6 abgelesen werden kann (s.o.).

Die Funktion $02 liest ein Zeichen ein, wobei im DX-Register wieder eine Null übergeben wird. Konnte das Zeichen korrekt empfangen werden, kann man dies am gelöschten Bit 7 im AH-Register ablesen, der ASCII-Code des Zeichens steht im Register AL zur Verfügung. Wie bei der Funktion $01 kann im Fehlerfall (Bit 7 im AH-Register gesetzt) die Ursache aus den Bits 0 bis 6 abgelesen werden.

Zwei weitere Funktionen dienen dazu, den Status der Schnittstelle und des Modems bzw. die Konfigurationsparameter zu erfragen. Dies sind die Funktion $03 des Interrupt $14, welche die gleichen Daten zurückliefert wie die Funktion $00, jedoch keine Initialisierung vornimmt, und die Funktion $19 des Interrupt $61, welche die gesetzten Parameter in der gleichen Form zurückliefert, wie sie gesetzt wurden (siehe Bild 1.16).

Neben diesen BIOS-Funktionen existieren noch zwei DOS-Funktionen zum Zugriff auf die serielle Schnittstelle: Durch die Funktion $03 des DOS-Interrupt $21 wird ein Zeichen eingelesen, über die Funktion $04 eines

ausgegeben. Ich rate von der Benutzung dieser Funktionen jedoch dringend ab: Der erste Nachteil besteht darin, daß keine Angaben über den Status zurückgeliefert werden. Man kann also nur hoffen, daß während der Übertragung alles korrekt ablief, hat aber keine Möglichkeit, es nachzuprüfen. Der zweite Nachteil besteht darin, daß die Routine zur Zeichenausgabe so lange wartet, bis das Modem (oder der Drucker etc.) bereit ist. Ist z.B. ein Drucker nicht angeschlossen, weil Papier fehlt, kann man solange warten, bis jemand Papier gekauft und eingelegt hat, eher wird die Funktion nicht in das Programm zurückkehren. Davon abgesehen sind beide Funktionen relativ langsam, so daß bei hohen Übertragungsraten ständig die Gefahr besteht, daß Zeichen beim Empfang überschrieben werden.

Wie schon oben erwähnt, stellt die serielle Schnittstelle als einziges Serienzubehör die Möglichkeit zur Verfügung, externe Interrupts auszulösen. Dafür muß man direkt auf die Register des UART zugreifen. Im Gegensatz zum PC liegen die Ports der Schnittstelle nicht ab der Adresse $03F8, sondern an einer Adresse, die man aus einer hierfür zuständigen BIOS-Variablen auslesen kann. Diese liegt nun ihrerseits definitiv an der Adresse $0040:$0017 und enthält die Startadresse der Ports. Wenn man diese Startadresse nun mit »Adresse« bezeichnet, findet man die folgenden Register des UART vor:

Adresse+0: Lesezugriff: Receive-Data-Register
Schreibzugriff: Transmitting-Holding-Register
Adresse+1: Interrupt-Enable-Register
Adresse+2: Interrupt-Identifikation-Register
Adresse+3: Line-Control-Register
Adresse+4: Modem-Control-Register
Adresse+5: Line-Status-Register
Adresse+6: Modem-Status-Register
Adresse+7: Scratch-Register

Wichtig für die Interrupt-Programmierung ist das Interrupt-Enable-Register: In ihm kann man festlegen, bei welchen Ereignissen ein Interrupt ausgelöst werden soll:

Bit 0 = 0: Kein Interrupt, wenn Daten empfangen wurden
Bit 0 = 1: Interrupt, wenn Daten empfangen wurden
Bit 1 = 0: Kein Interrupt, wenn Sendepuffer leer ist
Bit 1 = 1: Interrupt, wenn Sendepuffer leer ist
Bit 2 = 0: Schnittstellen-Status-Interrupt sperren
Bit 2 = 1: Schnittstellen-Status-Interrupt freigeben

Bit 3 = 0: Modemstatus-Interrupt sperren
Bit 3 = 1: Modemstatus-Interrupt freigeben
Bits 4–7 : Unbenutzt

Wenn man z.B. die Bits 0 und 1 setzt, kann man den seriellen Datentransfer völlig Interrupt-gesteuert vornehmen: Man schreibt das erste Zeichen in das Transmission-Holding-Register und fährt ganz normal im Programm fort. Wenn das Zeichen übertragen wurde, wird ein Interrupt ausgelöst und das nächste Zeichen kann zur Übertragung in den Puffer geschrieben werden. Auch um den Empfang braucht sich das Hauptprogramm nicht mehr zu kümmern: Falls ein Zeichen angekommen ist, wird ein Interrupt ausgelöst, dessen Service-Routine das Zeichen aus dem Receive-Data-Register abholen kann. Damit ist kein Polling mehr erforderlich, was kostbare Rechenzeit spart. Im Interrupt-Identification-Register ist immer das Bit gesetzt, das den Interrupt verursacht hat, so daß man bei mehreren zugelassenen Interrupts sehen kann, wodurch er verursacht wurde und entsprechend reagieren kann. Die Interrupts »Schnittstellen-Status« bzw. »Modemstatus« werden immer dann ausgelöst, wenn ein Bit innerhalb dieser Statusregister »Logisch 1« wird. Wenn Sie genau hinsehen, werden Sie denken, daß hier doppelt gemoppelt wurde, denn auch Bit 0 des Statusregisters der Schnittstelle steht für ein empfangenes Zeichen. Weshalb also noch der Extra-Interrupt? Nun, der Trick bei dieser Sache ist, daß man durch separates Verbieten dieses Interrupts bei gleichzeitigem Freigeben des Status-Interrupts ausschließlich dann Interrupts erhält, wenn ein Fehler aufgetreten ist (z.B. falsche Parität, Zeichen überschrieben etc.), nicht aber, wenn ein Zeichen empfangen wurde (was ja kein Übertragungsfehler ist). Um herauszufinden, weshalb ein Interrupt durch ein Status-Register ausgelöst wurde, liest man dieses einfach aus und prüft, welches Bit gesetzt ist.

Um nun Interrupts zu ermöglichen, sind zwei Dinge erforderlich: Zum einen muß man die Nummer des Interrupt, der ausgelöst werden soll, in das sogenannte Serial-Interrupt-Vektor-Register (SIVR), das sich an der Adresse $807F befindet, eintragen. Im Gegensatz zum PC besitzt die serielle Schnittstelle nämlich *keinen* festen Interrupt, so daß man auch innerhalb eines Programms durch mehrfaches Beschreiben dieses Registers den Interrupt wechseln kann. Der Vorteil: Man kann mehrere Interrupt-Service-Routinen installieren und je nach Bedarf eine davon aktivieren. Des weiteren muß man die Interrupts freigeben, indem man im Interrupt-Enable-Register die entsprechenden Bits setzt. Die Interrupt-Service-Routine muß zum korrekten Beenden des Interrupt unbedingt das

Interrupt-Identifikations-Register auslesen, auch wenn sie den Inhalt gar nicht benötigt (dies hat Hardware-spezifische Gründe).

Beispiel:

1. Interrupt definieren:
   $807F = 200 (Interrupt-Nummer 200 benutzen)
2. Interrupt-Vektor belegen:
   200x4 = Offset(Interrupt-Service-Routine)
   200x4+2 = Segment(Interrupt-Service-Routine)
3. Adresse UART holen:
   Adresse = $0040:$0017
4. Interrupts freigeben:
   Adresse+1 = 1 (Interrupt bei »Daten empfangen«)

Interrupt-Service-Routine:
.....
.....
Programmcode
.....
Dummyvariable = Adresse+2 (Interrupt-Identification-Register auslesen)
IRET

1.7 Die Echtzeituhr

Bislang konnte ich Ihnen leider nur von Inkompatibilitäten des Portfolio gegenüber dem PC berichten, die sich bei der Programmierung negativ auswirken. Doch der Winzling hält auch eine positive Überraschung bereit: eine batteriegepufferte Echtzeituhr, wie sie normalerweise nur im AT installiert ist. Der Vorteil gegenüber dem PC und XT besteht einfach darin, daß man die Uhr nicht bei jedem Neustart des Rechners neu setzen muß, sondern daß Datum und Uhrzeit, wenn sie einmal eingegeben wurden, für immer gespeichert bleiben (natürlich nicht, wenn die Batterien leer sind ...). Des weiteren kann man Alarmzeiten programmieren, bei denen ein Interrupt $4A ausgelöst wird. Dieses Verfahrens bedient sich z.B. der Terminplaner; dazu jedoch später.

Alle Eingaben der Echzeituhr müssen im sogenannten BCD-Format vorgenommen werden, was mit der Hardware der Uhr zusammenhängt. Dies bedeutet, daß eine Dezimalzahl (0 bis 9) immer durch eine Gruppe von

4 Bit repräsentiert wird. In der Praxis bedeutet dies, daß von 0 bis 9 keine Änderung gegenüber dem normalen Zahlenformat festzustellen ist. Wenn jedoch die Zahl 9 überschritten wird, bedeutet dies im Dezimalformat, daß eine neue Stelle hinzugefügt werden muß, also auch eine neue Vierergruppe Bits »aufgemacht« werden muß, wobei dann wieder ganz normal bei der Zahl 0 begonnen wird. Die folgende Tabelle stellt die BCD-Zahlen von 0 bis 100 dar:

BCD-Codes von 0 bis 100

Dezimal	BCD-Code (Binär)	BCD-Code (Dezimal)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	0001 0000	16
11	0001 0001	17
..	....	..
19	0001 1001	25
20	0010 0000	32
21	0010 0001	33
..	....	..
49	0100 1001	73
50	0101 0000	80
51	0101 0001	81
..	....	...
96	1001 0110	150
97	1001 0111	151
98	1001 1000	152
99	1001 1001	153
100	0001 0000 0000	256

Für die Programmierung der Echtzeituhr stellt das Portfolio-BIOS insgesamt sechs Funktionen des Interrupt $1A zur Verfügung.

Funktion $02: Uhrzeit auslesen
Ausgabe:
Wenn Carry-Flag = 0: CH = Stunde, CL = Minute, DH = Sekunde (im BCD-Format!)
Wenn Carry-Flag = 1: Fehler (Batterie ist leer)

Funktion $03: Uhrzeit setzen
Eingabe:
CH = Stunde, CL = Minute, DH = Sekunde, DL = Kalender (DL=1: Es ist Sommerzeit, DL=0:Es ist keine Sommerzeit)

Funktion $04: Datum auslesen
Ausgabe:
Wenn Carry-Flag = 0: CH = Jahrhundert (19 oder 20), CL = Jahr, DH = Monat, DL = Tag (im BCD – Format!)
Wenn Carry-Flag = 1: Fehler (Batterie ist leer)

Funktion $05: Datum setzen
Eingabe:
CH = Jahrhundert (19 oder 20), CL = Jahr, DH = Monat, DL = Tag (im BCD-Format)

Funktion $06: Alarmzeit setzen
Eingabe:
CH = Stunde, CL = Minute, DH = Sekunde (im BCD-Format)
Ausgabe:
Wenn Carry-Flag = 0: Alarmzeit wurde programmiert
Wenn Carry-Flag = 1: Fehler (Batterie leer oder bereits programmierte Alarmzeit)

Funktion $07: Alarmzeit löschen
Eingabe: Keine

Da immer nur ein Alarm gleichzeitig programmiert werden kann, muß bei einer Verschiebung einer Alarmzeit diese zunächst mit der Funktion $07 gelöscht werden, bevor mit Hilfe der Funktion $06 eine neue Zeit programmiert werden kann. Die Alarmzeit bezieht sich dabei immer auf den aktuellen Tag. Bei Erreichen der Alarmzeit wird ein Interrupt $4A ausgelöst. Die Interrupt-Service-Routine besteht nach dem Starten des Portfolio nur aus dem Befehl IRET, so daß sofort ins aufrufende Programm zurückgekehrt wird, ohne daß irgendetwas passieren würde. Da man im Normalfall aber auf einen Alarm reagieren möchte, muß man den Interrupt-Vektor auf eine eigene Interrupt-Routine verbiegen, die dann auf den Alarm reagieren kann. Das folgende Mini-Beispielprogramm setzt zunächst

Datum und Uhrzeit und programmiert dann einen Alarm für eine Minute später. Nach dem Programmstart sehen Sie nur einen blinkenden Cursor, da das Programm auf einen Tastendruck wartet. Nach einer Minute jedoch erfolgt eine Bildschirmausgabe, die durch den WRITE-Befehl innerhalb der Interrupt-Prozedur vorgenommen wird. Deshalb wurde auch ganz zum Anfang der Interrupt-Vektor $4A auf diese Interrupt-Routine umgeleitet. Nach einem Tastendruck wird das Programm beendet. Um das folgende Programm ablaufen zu lassen, ist die CRT-Unit für den Portfolio aus dem Kapitel 2.4 erforderlich. Diese Unit ist auf der Begleitdiskette enthalten.

```
PROGRAM ECHTZEITUHR;

{$M 3000,0,0}

USES portcrt,dos;

var reg : registers;
intofs : word absolute $0000:$0128;
intseg : word absolute $0000:$012A;

PROCEDURE alarm;
interrupt;
BEGIN
  writeln(' * * * * A L A R M  ! ! ! ! ! * * * * ');
END;

BEGIN

  intseg := Seg(alarm);        {Interrupt-Vektor $4A neu setzen}
  intofs := Ofs(alarm);

  {Datum auf 27.9.1990 setzen}

  reg.ah := 5;
  reg.ch := 25;     {19 im BCD-Format}
  reg.cl := 144;    {90 im BCD-Format}
  reg.dh := 9;      { 9 im BCD-Format}
  reg.dl := 39;     {27 im BCD-Format}
  Intr($1A,reg);

  {Uhrzeit auf 9:41:00 setzen}

  reg.ah := 3;
  reg.ch := 9;     { 9 im BCD-Format}
  reg.cl := 64;    {40 im BCD-Format}
  reg.dh := 0;     { 0 im BCD-Format}
  reg.dl := 0;     {Keine Sommerzeit}
  Intr($1A,reg);

  {Alarmzeit löschen}
```

```
  reg.ah := 7;
  Intr($1A,reg);

  {Alarmzeit auf 9:42:00 setzen}

  reg.ah := 6;
  reg.ch := 9;      { 9 im BCD-Format}
  reg.cl := 65;     {41 im BCD-Format}
  reg.dh := 0;      { 0 im BCD-Format}
  Intr($1A,reg);

  REPEAT
  UNTIL Keypressed;

END.
```

Kapitel 2: Programmierung in Turbo Pascal und Turbo Assembler

2.1 Turbo-Pascal-Programme auf dem Portfolio

Wenn man der Werbung von Atari Glauben schenkt, müßte an sich jedes für den PC entwickelte Programm auch auf dem Portfolio laufen, falls der Arbeitsspeicher dafür ausreicht. In Kapitel 1 wurde jedoch deutlich, daß wesentliche Hardware-Unterschiede zum PC bestehen und dieses Wunschdenken daher nicht der Realtität entspricht. Besonders bitter trifft dies die Pascal-Programmierer unter ihnen, die besonders mit drei Problemen konfrontiert werden:

- Durch einen falschen Eintrag im Kopf der Pascal-.EXE-Datei lassen sich viele Programme erst gar nicht starten.
- Die CRT- und GRAPH-Units laufen nicht, sondern lassen den Portfolio abstürzen.
- Durch das Fehlen des Interrupt $28 laufen keine TSR-Programme herkömmlicher Art.

Nun, Probleme sind dazu da, daß man sie löst. In den folgenden Abschnitten werden wir Schritt für Schritt eine Pascal-Programmierung auch auf dem Portfolio möglich machen. Die meisten Programme enthalten dabei Assembler-Routinen, die mit dem Turbo Assembler entworfen wurden und dann in das Pascal-Programm eingebunden wurden. Bei Turbo-Assembler-Programmen gibt es keinerlei Schwierigkeiten auf dem Portfolio, da ihr Code ja zu 100% von einem selbst entworfen wird und nicht durch vorgefertigte Compiler-Routinen ergänzt wird, die bei Turbo Pascal im Endeffekt für die Inkompatibilitäten verantwortlich sind.

2.2 Einstellungen der Compiler

Für die Turbo-Pascal-Programmierung werden heute praktisch nur noch die Compiler der Versionen 4.0 und 5.0 bzw. 5.5 eingesetzt. Bevor man sich Gedanken über das eigentliche Programm macht, sollte man für den Portfolio sinnvolle Compiler-Voreinstellungen wählen. Bei diesem Rechner müssen die Programme vor allen Dingen unter dem Gesichtspunkt der Minimierung des Speicherbedarfs erstellt werden. Dies bedeutet, daß jeder zusätzliche Prüfcode, Debug-Informationen und Symboltabellen in Portfolio-Programmen fehl am Platz ist, zumal man auf dem Portfolio sowieso nicht debuggen kann. Im *OPTIONS*-Menü unter dem Menüpunkt *COMPILER* sollten daher für die Compiler-Version 4.0 folgende Einstellungen vorgenommen werden:

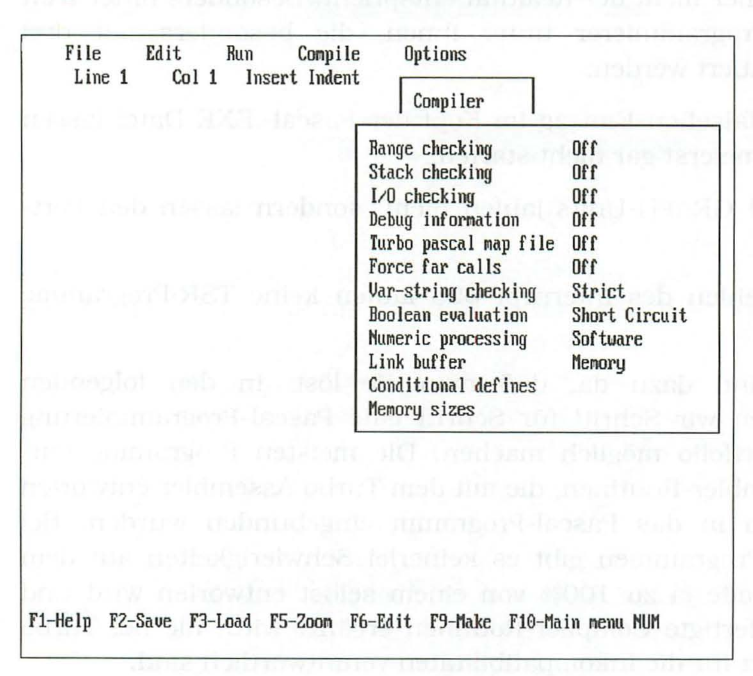


Bild 2.1: Compiler-Einstellungen für Turbo Pascal 4.0

Entsprechend sind für die Versionen 5.0 und 5.5 diese Einstellungen sinnvoll:

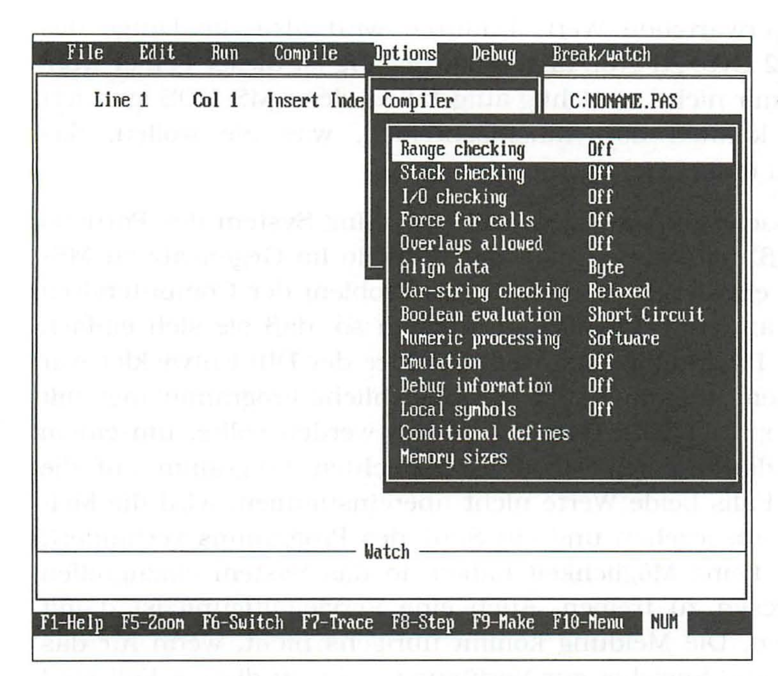


Bild 2.2:Compiler-Einstellungen für Turbo Pascal 5.0 und 5.5

Besonders durch den Wegfall des zusätzlichen Prüfcodes für die Punkte *Range-Checking* und *Stack-Checking* wird eine Unmenge an Speicherplatz gespart. Informationen für den Turbo Debugger sind ohnehin wertlos, da sich dieser auf dem Portfolio nicht installieren läßt. Ich kenne Programme, die bei einer Kompilierung mit den obigen Einstellungen mehr als 50 % kürzer wurden als bei einer Kompilierung mit den Standardvorgaben und erst dadurch auf dem Portfolio lauffähig wurden.

2.3 Eintrag der korrekten Programmlänge in den Kopf der EXE-Datei

Wenn Sie sich einmal mit einem Diskettenmonitor die ersten Bytes einer EXE-Datei anzeigen lassen, finden Sie als erstes Wort den Eintrag $54AD. Dieses ist eine Kennzeichnung, durch die eine Datei als Typ *EXE* gekennzeichnet wird. Die nächsten beiden Wörter enthalten nun die Dateilänge, die wie folgt berechnet wird:

Dateilänge = zweites Wort x 512 + erstes Wort

Der Turbo-Pascal-Compiler enthält bereits seit Version 4.0 einen Fehler: Er trägt in das erste Wort einen Wert ein, der genau um eins größer ist als

der eigentlich zu erwartende Wert. Dadurch wird also die Länge des Programms um 512 Byte zu groß angesetzt. Bislang ist dieser Fehler wohl niemandem (auch mir nicht) so richtig aufgefallen, denn MS-DOS ignoriert den Eintrag. Sie können dort hineinschreiben, was Sie wollen, das Programm läßt sich immer richtig starten.

Anders sieht die Sache nun mit dem DIP-Operating-System des Portfolio aus. Zunächst muß man wissen, daß der Portfolio im Gegensatz zu MS-DOS zu einer Zeit entwickelt wurde, als das Problem der Computerviren dramatische Züge annahm. Viele Viren arbeiten so, daß sie sich einfach an das bestehende Programm anhängen. Die Idee der DIP-Entwickler war nun die, daß vor dem Programmstart die tatsächliche Programmlänge mit dem Eintrag im Kopf der EXE-Datei verglichen werden sollte, um einem verlängerten und damit potenziell virenverseuchten Programm auf die Spur zu kommen. Falls beide Werte nicht übereinstimmen, wird die Meldung *EXEC-Fehler* ausgegeben und ein Start des Programms verhindert, wodurch die Viren keine Möglichkeit haben, in das System einzugreifen und dort ihr Unwesen zu treiben. Auch eine Vervielfältigung ist damit unmöglich geworden. Die Meldung kommt übrigens nicht, wenn für das Programm nicht genug Speicher zur Verfügung steht, in diesem Fall wird die Meldung *Nicht genug Speicher* ausgegeben.

Sicherlich ist dieses Anti-Viren-Betriebssystem in der heutigen Zeit sinnvoll. Das Problem besteht jedoch darin, daß es nicht zwischen tatsächlich virenverseuchten Programmen und solchen unterscheiden kann, die im Grunde genommen harmlos sind (wie unsere Pascal-Programme) und trotzdem einen falschen Eintrag im Kopf der EXE-Datei aufweisen. Interessanterweise enthält das Betriebssystem jedoch noch einen Fehler, der den Start mancher Programme erlaubt, obwohl der Eintrag nicht mit der tatsächlichen Programmlänge übereinstimmt. Hier hilft nur ausprobieren.

Wenn Sie aber die Meldung *EXEC-Fehler* erhalten sollten, müssen Sie einfach nur die tatsächliche Programmlänge in die beiden Wörter im Kopf eintragen, und schon läuft Ihr Programm auf dem Portfolio. Die Korrektur kann man umständlich per Hand durchführen oder einem Programm überlassen. Da letzteres sicherlich bequemer ist, habe ich das Korrekturprogramm PORTMAKER entworfen, welches die Bytes einer Datei zählt und anschließend den Eintrag im Kopf der Datei korrigiert.

```
PROGRAM Portmaker;
{written by Frank Riemenschneider
            Postfach 730309
            3000 Hannover 71}


{$I-}
{$M 4000,2000,2000}
USES dos,crt;
VAR
zaehler : longint;
f : file;
high : word;
low : word;
name : string;
dummy : byte;
frage : char;
CONST
 fullstern : string[80] =
  '**********************************************'+
  '**********************************';
 randstern : string[1] = '*';

{— Texteingabe: Bildschirmeingabe eines Textes  —}
{— Eingabe: StringVARiable, Koordinatenpaar     —}
{— x,y (0..79,0..24), maximale Länge,           —}
{— Ausgabe: Texteingabe                         —}

FUNCTION Eingabe(x,y,maxlaenge : byte) : string;
VAR taste,laenge,i : byte;
ss : string;
BEGIN
  GotoXY(x,y);
  ss := '';
  laenge := 0;
  REPEAT
    taste := Ord(ReadKey);                {ASCII-Code holen}
    CASE taste of

          32..255 : BEGIN
                      IF laenge < maxlaenge THEN BEGIN
                        inc(x);
                        inc(laenge);
                        write(chr(taste));
                        ss := ss + chr(taste);
                      END;
                     END;

    {Hier BEGINnen, wenn Backspace gedrückt}
```

```
        8   : BEGIN
                IF laenge > 0 THEN BEGIN
                  dec(x);
                  dec(laenge);
                  GotoXY(x,y);              {Letztes Zeichen löschen}
                  write(' ');
                  GotoXY(x,y);
                  ss := copy(ss,1,Length(ss)-1);
                END;
              END;
    END;
  UNTIL taste = 13;          {Abschluß mit RETURN}
  Eingabe := ss;
  writeln;
END;

{— Standardbild: Bildschirm aufbauen                        —}
{— Eingabe: keine  Ausgabe: Keine                           —}

PROCEDURE Standardbild;
VAR i : byte;
BEGIN
   ClrScr;
   write(fullstern);
   GotoXY(1,3);
   write('************************** M & T  P O R T M A K E R **************************');
   GotoXY(1,5);
   write('*********** (C)opyright 1990 by Markt & Technik Aktiengesellschaft ***********');
   GotoXY(1,7);
   write('***************** written 1989,1990 by Frank Riemenschneider *****************');
   GotoXY(1,9);
   write(fullstern);
   GotoXY(0,0);
   For i := 1 to 22 DO BEGIN
     GotoXY(1,i);
     write(randstern);
     GotoXY(80,i);
     write(randstern);
   END;
   write(fullstern);
   GotoXY(4,11);
   write('PORTMAKER macht Turbo-Programme (Pascal 4.0, 5.0 und 5.5, C 2.0 und C++,');
   GOTOXY(4,13);
   write('Assembler 1.X und 2.0) auf dem Portfolio lauffähig, indem die wirkliche');
   GOTOXY(4,15);
   write('Programmlänge in den Kopf der .EXE-Datei eingetragen wird.');
   GOTOXY(4,17);
   write('Bitte geben Sie Pfad- und Dateiname der zu korrigierenden Datei ein:');
END;
```

```
{——— Hauptprogramm ——————}

BEGIN
 Standardbild;
 REPEAT
   GOTOXY(4,19);
   write('Dateiname : <                                                    >');
   name := Eingabe(17,19,50);

   {Falls das Suffix '.exe' nicht mit eingegeben wurde, wird dies angehängt }

   IF Copy(name,length(name)-3,1) <> '.'THEN name := name + '.EXE';
   GotoXY(4,21);
   Randomize;
   Assign(f,name);
   Reset(f,1);
   IF IOResult = 0 THEN BEGIN
     zaehler := 0;
     WHILE ((not EOF(f)) and (IOResult = 0)) DO BEGIN
       blockread(f,dummy,1);

       {Datenbytes zählen,alle 500 Bytes eine Meldung ausgeben}

       inc(zaehler);
       IF Trunc(zaehler/500) = (zaehler/500) THEN BEGIN
         GotoXY(4,21);
         write('                                                    ');
         GotoXY(4+Trunc(Random*50),21);
         write('Arbeite...');
         GotoXY(4,21);
       END;
     END;
     IF EOF(f) THEN BEGIN

       {Eintrag für Kopf der Datei berechnen und hineinschreiben}

       high := Trunc(zaehler/512);
       low := zaehler - high*512;
       Seek(f,2);
       IF IOResult = 0 THEN blockwrite(f,low,2);
       IF IOResult = 0 THEN blockwrite(f,high,2);
       IF IOResult = 0 THEN BEGIN
         write('Die Datei wurde korrigiert. Weitere Dateien umwandeln (J/N)?');
       END
       ELSE BEGIN
         write('Beim Schreiben trat ein Fehler auf. Weitere Dateien umwandeln (J/N)?');
       END;
     END
     ELSE BEGIN
       write('Beim Lesen trat ein Fehler auf. Weitere Dateien umwandeln (J/N)?');
     END;
     close(f);
   END
```

```
    ELSE BEGIN
      write('Fehler beim Öffnen der Datei. Weitere Dateien umwandeln (J/N)?');
    END;
    REPEAT
      frage := UpCase(ReadKey);
    UNTIL ((frage = 'J') or (frage = 'N'));
    GotoXY(4,21);
    write('                                                            ');
  UNTIL (frage = 'N');
  ClrScr;
END.
```

Nach dem Programmstart (bitte auf dem PC!) werden Sie einfach nach dem Namen der zu korrigierenden Datei gefragt, wobei Sie das Suffix .EXE optional angeben oder weglassen können. Anschließend wird die Datei bearbeitet. Sie werden nun gefragt, ob noch weitere Dateien bearbeitet werden sollen. Nach der Eingabe [N] wird das Programm beendet.

Bitte beachten Sie, daß der Portfolio nach dem EXEC-Fehler manchmal ein wenig durcheinander ist und sich auch das korrigierte Programm dann nicht starten läßt. Hier hilft ein Hardware-Reset weiter, wonach die Bausteine neu initialisiert werden. Anschließend können Sie das Programm ohne Probleme starten.

2.4 Eine CRT-Unit für den Portfolio

Das größte Problem für Portfolio-Programmierer besteht sicherlich darin, daß die CRT-Unit von Turbo Pascal überhaupt nicht läuft, sondern den Rechner zum Absturz bringt. Damit fehlen die wichtigen Routinen zur Bildschirmprogrammierung, wie z.B. Setzen des Cursors, Bildschirmlöschen etc.

Ich habe deshalb eine eigene auf den Portfolio zugeschnittene Unit programmiert, die alle Befehle der Original-Unit ersetzt, soweit sie auf dem Portfolio sinnvoll anzuwenden sind. Dabei wurden nicht nur die Originalnamen der Befehle verwendet, sondern soweit möglich auch ein identisches Aufrufformat, so daß die meisten PC-Programme gar nicht erst verändert werden müssen. Es ist lediglich die Original-CRT-Unit durch die neue mit dem Namen PORTCRT zu ersetzen.

Zusätzlich zu diesen Befehlen wurden noch einige Portfolio-spezifische hinzugefügt, insbesondere, um die diversen Bildschirmmodi ansprechen

zu können. Bevor ich Ihnen die Befehle im einzelnen vorstelle, hier zunächst das Listing der PORTCRT-Unit:

```
UNIT portcrt;
{written by Frank Riemenschneider
            Postfach 730309
            3000 Hannover 71}

{$L a:crtasm.obj}

INTERFACE

CONST
portfolio = 1;
statisch = 0;
dynamisch = 2;
oben = 1;
unten = 2;                  {Konstanten für Bildschirmsteuerung}
links = 3;
rechts =4;

VAR
xmax,ymax : byte;

PROCEDURE Sound(frequenz,dauer : word);
FUNCTION KeyPressed : boolean;
FUNCTION ReadKey : Char;
PROCEDURE Delay(zeit : word);
PROCEDURE ClrScr;
PROCEDURE Window(xl,yl,xr,yr : byte);
PROCEDURE Textmode(mode : byte);
PROCEDURE GotoXY(x,y : byte);
PROCEDURE DelLine(y : byte);
PROCEDURE InsLine(y : byte);
FUNCTION StoreBild(x1,y1,x2,y2 : word) : pointer;
PROCEDURE LoadBild(p : pointer);
PROCEDURE ScrollScreen(off,richtung : byte);
PROCEDURE MoveScreen(xpos,ypos : byte);
FUNCTION Portmenue(titel : string; xpos,ypos,anzahl,breite : byte;
                   VAR punkte) : byte;
FUNCTION WhereX : byte;
FUNCTION WhereY : byte;
FUNCTION GetScreenMode : byte;
FUNCTION GetScreenPos : word;

IMPLEMENTATION

USES dos;
```

```
VAR reg: registers;
    saveexit : pointer;
    int_16 : pointer absolute $0000:$0040;
    int_10 : pointer absolute $0000:$0028;
    int_06 : word absolute $0000:$0018;
    modus,speicher : byte;
    xlo,xru,ylo,yru : byte;
    retint : pointer;

{$F+}
PROCEDURE NewInt; EXTERNAL;
PROCEDURE Save(s,o,size,x1,y1,x2,y2 : word); external;
FUNCTION Load(s,o:word) : word; external;
{$F-}

{— Sound: Schaltet Lautsprecher ein                           —}
{— Eingabe: Frequenz in Herz, Tondauer                        —}
{— Ausgabe: Keine                                             —}

PROCEDURE Sound(frequenz,dauer : word);
VAR reg : registers;
i : byte;
CONST
freq : array[1..25] of word = (2489,2349,2217,2093,1975,1865,1760,1661,1568,1480,
                               1397,1318,1245,1175,1109,1046,988,932,880,831,784,
                               740,699,659,622);
wert : array[1..25] of byte = (7,6,$2F,$25,5,4,$3F,$2C,$3E,$0E,$3D,$3C,
                               $3B,$29,$3A,$39,$38,$37,$36,$35,$34,$33,
                               $32,$31,$30);
BEGIN
  i := 1;
  WHILE ((freq[i] > frequenz) and (i<= 25)) DO BEGIN
    inc(i);
  END;
  reg.dl := wert[i];
  reg.ah := 22;
  reg.cx := dauer;
  Intr(97,reg);
END;

{— Delay: Wartet eine bestimmte Zeit                          —}
{— Eingabe: Zeit in Millisekunden                             —}
{— Ausgabe: Keine                                             —}

PROCEDURE Delay(zeit : word);
VAR i,j : word;
BEGIN
 FOR i:= 0 to zeit DO BEGIN
  FOR j:= 0 to 55 DO BEGIN
  END;
 END;
END;
```

```
{— KeyPressed: Prüft, ob Taste gedrückt          —}
{— Eingabe: Keine                                —}
{— Ausgabe: Ja/Nein   (True/False)               —}

FUNCTION Keypressed : boolean;
VAR reg : registers;
BEGIN
  reg.ah := 1;
  Intr(22,reg);
  IF (reg.flags and 64) <> 0 THEN BEGIN
    Keypressed := False;
  END
  ELSE BEGIN
    KeyPressed := True;
  END;
END;

{— ReadKey: Holt Zeichen von Tastatur            —}
{— Eingabe: Keine                                —}
{— Ausgabe: ASCII/Scan-Code                      —}

FUNCTION ReadKey : char;
VAR reg : registers;
BEGIN
  IF speicher = 0 THEN BEGIN
    REPEAT
      reg.ah := 1;
      Intr(22,reg);
    UNTIL (reg.flags and 64) = 0;
    reg.ah := 0;
    Intr(22,reg);
    ReadKey := chr(reg.al);
    IF reg.al = 0 THEN BEGIN
      speicher := reg.ah;
    END;
  END
  ELSE BEGIN
    ReadKey := chr(speicher);
    speicher := 0;
  END;
END;

{— WhereX: Holt X-Position Textcursor            —}
{— Eingabe: Keine                                —}
{— Ausgabe: X-Position                           —}

FUNCTION WhereX : byte;
VAR reg : registers;
BEGIN
    reg.ah := 3;
    reg.bh := 0;                    {Interrupt hex. 10, Funktion 3}
    intr(16,reg);                   {in PC INTERN auf Seite 922}
    WhereX := reg.dl - xlo;
END;
```

```
{— WhereY: Holt Y-Position Textcursor                    —}
{— Eingabe: Keine                                         —}
{— Ausgabe: Y-Position                                    —}

FUNCTION WhereY : byte;
VAR reg : registers;
BEGIN
    reg.ah := 3;
    reg.bh := 0;                 {Interrupt hex. 10, Funktion 3}
    intr(16,reg);                {in PC INTERN auf Seite 922}
    WhereY := reg.dh - ylo;
END;

{— GotoXY: Positioniert Textcursor                        —}
{— Eingabe: Koordinatenpaar x,y (0..39,0..7)              —}
{— Ausgabe: Keine                                         —}

PROCEDURE GotoXY(x,y : byte);
VAR reg : registers;
BEGIN
  IF ((x >=0) and (x <= xru)  and (y>=0) and (y<= yru)) THEN BEGIN
    reg.ah := 2;
    reg.bh := 0;
    reg.dl := x+xlo;
    reg.dh := y+ylo;             {Interrupt hex. 10, Funktion 2,}
    intr(16,reg);                {in PC INTERN auf Seite 922}
  END;
END;

{— ClrScr: Löscht Bildschirmfenster                       —}
{— Eingabe: Modus                                         —}
{— Ausgabe: Keine                                         —}

PROCEDURE ClrScr;
VAR reg: registers;
BEGIN
  reg.ah := 6;
  reg.al := 0;
  reg.ch := ylo;
  reg.cl := xlo;
  reg.dh := yru;
  reg.dl := xru;
  reg.bh := 15;
  Intr(16,reg);
  GotoXY(0,0);
END;

{— Window: Definiert Fenster                              —}
{— Eingabe: linke obere Ecke, rechte untere Ecke          —}
{— Ausgabe: Keine                                         —}

PROCEDURE Window(xl,yl,xr,yr : byte);
```

```
BEGIN
  IF ((xl >= 0) and (xl<=xmax) and (xr>=0) and (xr<=xmax) and
    (yl >= 0) and (yl<=ymax) and (yr>=0) and (yr<=ymax)) THEN BEGIN
       IF ((xl>0) or (yl>0) or (xr<xmax) or (yr<ymax))
         THEN BEGIN
          int_16 := @NewInt;  {Neuen Interrupt-Vektor 10h setzen}
       END
       ELSE BEGIN
          int_16 := retint;   {Interrupt-Vektor 10h wieder auf alten Wert}
       END;
       xlo := xl;
       ylo := yl;
       xru := xr;
       yru := yr;
       GotoXY(0,0);
  END;
END;

{— Textmode: Schaltet Textmodus + löscht Bildschirm    —}
{— Eingabe: Modus                                       —}
{— Ausgabe: Keine                                       —}

PROCEDURE Textmode(mode : byte);
VAR reg : registers;
BEGIN
 IF ((mode >= 0) and (mode <=2)) THEN BEGIN
  reg.ah := $0E;
  reg.al := 1;
  reg.dl := mode;
  Intr(97,reg);
  modus := mode;
  CASE mode of
   0,2 : BEGIN
          xmax := 79;
          ymax := 24;
         END;
     1 : BEGIN
          xmax := 39;
          ymax := 7;
         END;
  END;
  Window(0,0,xmax,ymax);
  ClrScr;
 END;
END;

{— MoveScreen: Bewegt Fensterauschnitt                 —}
{— Eingabe: X- und Y-Koordinate der linken, oberen Ecke  —}
{— Ausgabe: Keine                                        —}

PROCEDURE MoveScreen(xpos,ypos : byte);
VAR reg : registers;
```

```
BEGIN
  IF ((xpos+39<=xmax) and (ypos+7<=ymax) and
      (modus = 0) or (modus =2)) THEN BEGIN
     reg.ah := $10;
     reg.al := 1;
     reg.dl := xpos;
     reg.dh := ypos;
     Intr(97,reg);
  END;
END;

{— GetScreenPos: Holt Fenster-Position                —}
{— Eingabe: Keine                                     —}
{— Ausgabe: Fenster-Position                          —}

FUNCTION GetScreenPos : word;
VAR reg : registers;
BEGIN
  reg.ah := $10;
  reg.al := 0;
  Intr(97,reg);
  GetScreenPos := reg.dx;
END;

{— ScrollScreen: Bewegt Fensterauschnitt              —}
{— Eingabe: Offset (Zeilen oder Spalten), Richtung    —}
{— Ausgabe: Keine                                     —}

PROCEDURE ScrollScreen(off,richtung : byte);
VAR reg : registers;
pos : word;
xvirt,yvirt : byte;
BEGIN
  pos := GetScreenPos;
  xvirt := pos and 255;
  yvirt := pos shr 8;
  IF (((modus = 0) or (modus =2))
    and (((richtung = 1) and ((yvirt - off) >=0))
    or ((richtung = 2) and ((yvirt + off + 7) <= ymax))
    or ((richtung = 3) and ((xvirt - off) >=0))
    or ((richtung = 4) and ((xvirt + off + 39) <= xmax)))) THEN BEGIN
     reg.ah := $11;
     reg.al := off;
     reg.dl := richtung;
     Intr(97,reg);
     CASE richtung of
       1 : yvirt := yvirt - off;
       2 : yvirt := yvirt + off;
       3 : xvirt := xvirt - off;
       4 : xvirt := xvirt + off;
     END;
  END;
END;
```

```
{— GetScreenMode: Holt Bildschirmmodus                    —}
{— Eingabe: Keine                                         —}
{— Ausgabe: Bildschirmmodus                               —}

FUNCTION GetScreenMode : byte;
VAR reg : registers;
BEGIN
  reg.ah := $0E;
  reg.al := 0;
  Intr(97,reg);
  GetScreenMode := reg.dl;
END;


{— DelLine: Löscht Bildschirmzeile                      —}
{— Eingabe: Keine                                       —}
{— Ausgabe: Keine                                       —}

PROCEDURE DelLine(y:byte);
VAR reg: registers;
BEGIN
  IF ((y>=0) and (y+ylo<=yru)) THEN BEGIN
    IF y+ylo < yru THEN BEGIN
      reg.al := 1;
      reg.ch := y+ylo;
    END
    ELSE BEGIN
      reg.al := 0;
      reg.ch := yru;
    END;
    reg.ah := 6;
    reg.cl := xlo;
    reg.dh := yru;
    reg.dl := xru;
    reg.bh := 15;
    Intr(16,reg);
  END;
END;

{— InsLine: Fügt Bildschirmzeile ein                    —}
{— Eingabe: Keine                                       —}
{— Ausgabe: Keine                                       —}

PROCEDURE InsLine(y:byte);
VAR reg: registers;
BEGIN
  IF ((y>=0) and (y+ylo<=yru)) THEN BEGIN
    IF y+ylo < yru THEN BEGIN
      reg.al := 1;
      reg.ch := y+ylo;
    END
    ELSE BEGIN
      reg.al := 0;
      reg.ch := yru;
```

```
    END;
    reg.ah := 7;
    reg.cl := xlo;
    reg.dh := yru;
    reg.dl := xru;
    reg.bh := 15;
    Intr(16,reg);
  END;
END;

{— StoreBild: Rettet Bildschirmbereich auf Heap        —}
{— Eingabe: Linke obere (X1,Y1) und rechte untere       —}
{—          (X2,Y2) Ecke des zu rettenden Bereichs      —}
{— Ausgabe: Pointer auf Heap                            —}

FUNCTION StoreBild(x1,y1,x2,y2 : word) : pointer;
VAR p :pointer;
size : word;
BEGIN
  IF ((x1 >= 0) and (x2 <= xmax) and (y1 >=0) and (y2<=ymax)) THEN BEGIN
    size := 10+((2*(x2-x1+1))*(y2-y1+1));
    GetMem(p,size);
    save(Seg(p),Ofs(p),size,x1,y1,x2,y2);
    storebild := p;
  END
  ELSE BEGIN
    storebild := NIL;
  END;
END;

{— LoadBild: Kopiert Daten in Bildschirm               —}
{— Eingabe: Pointer auf Daten                           —}
{— Ausgabe: Keine                                       —}

PROCEDURE LoadBild(p:pointer);
VAR size : word;
BEGIN
  size := Load(Seg(p),Ofs(p));
  FreeMem(p,size);
END;

{— Portmenue: Aufbau und Auswertung eines Portfolio-Menüs     —}
{— Eingabe: Menüstruktur                                       —}
{— Ausgabe: Gewählter Menüpunkt (0=Abbruch)                    —}

FUNCTION Portmenue(titel : string; xpos,ypos,anzahl,breite : byte;
                   VAR punkte) : byte;
VAR p1 : pointer;
i : byte;
s : string;
code : byte;
punkt : byte;
retx,rety,xl,xr,yo,yu : byte;
flag : boolean;
```

```
TYPE name = array[1..23] of string;
CONST
leer : string[80] = '                                        '+
                    '                                        ';
linie : string[80] = '════════════════════════════════════════'+
                     '════════════════════════════════════════';

BEGIN
 IF ((xlo+xpos+breite <= xru+1) and (ylo+ypos+anzahl < yru) and (breite*(anzahl-2)<=255))
THEN BEGIN
  s := chr(201)+copy(linie,1,breite-2)+chr(187)+leer;
  IF titel <>'' THEN BEGIN
    titel := Copy(titel,1,breite-2);
    punkt := Trunc((breite-length(titel))/2);
    FOR i:= 1 to length(titel) DO BEGIN
      s[punkt+i] := titel[i];
    END;
  END;
  FOR i:= 1 to anzahl DO BEGIN
    insert(chr(186)+chr(32)+Copy(name(punkte)[i],1,breite-3),s,1+(breite*i));
    insert(chr(186)+leer,s,breite*(i+1));
  END;
  s := copy(s,1,breite*(anzahl+1))+chr(200)+copy(linie,1,breite-2)+chr(188);
  xl := xlo;
  xr := xru;
  yo := ylo;
  yu := yru;
  retx := WhereX;
  rety := WhereY;
  pl := StoreBild(xpos,ypos,xpos+breite-1,ypos+anzahl+1);
  window(xpos,ypos,xpos+breite-1,ypos+anzahl+1);
  write(s);
  punkt := 1;
  flag := false;
  REPEAT
    GotoXY(2,punkt);
    code := Ord(ReadKey);
    IF code<> 0 THEN BEGIN
      CASE code of
         13: flag := true;          {RETURN}
         27: BEGIN
               flag:= true;         {ESCAPE}
               punkt := 0;
             END;
      32..155 :  BEGIN
                   FOR i:= 1 to anzahl DO BEGIN         {Buchstabe, Zahl etc.}
                     s := Copy(name(punkte)[i],1,1);
                     IF ((s[1] = chr(code)) or (s[1] = Upcase(chr(code)))) THEN BEGIN
                        punkt := i;
                        i := anzahl;
                        flag := true;
                     END;
                   END;
                 END;
```

```
      END;
    END
    ELSE BEGIN
      code := Ord(ReadKey);
      CASE code of
         $50: IF punkt < anzahl THEN inc(punkt);          {Cursor runter}
         $48: IF punkt > 1 THEN dec(punkt);               {Cursor rauf}
      END;
    END;
  UNTIL (flag=true);
  xlo := xl;
  xru := xr;
  ylo := yo;
  yru := yu;
  LoadBild(pl);
  GotoXY(retx,rety);
  portmenue := punkt;
 END
 ELSE BEGIN
  portmenue := 0;
 END;
END;

{$F+}
PROCEDURE NewExit;                {Neuer Exit-Code der Unit:}
BEGIN
  ExitProc := SaveExit;
  int_16 := retint;               {Interrupt-Vektor 10h wieder auf alten Wert}
END;
{$F-}

{----------Beginn der Initialisierungsroutine------------}

BEGIN
 speicher := 0;
 retint := int_16;                {Interrupt 10h retten}
 int_10 := retint;
 int_06 := Dseg;
 reg.ah := 0;                     {Portfolio-Interrupt initialisieren}
 Intr(97,reg);
 reg.ah := $0E;                   {Screen-Modus holen}
 reg.al := 0;
 Intr(97,reg);
 modus := reg.dl;
 Textmode(modus);
 MoveScreen(0,0);                 {Fenster in linke obere Ecke bringen}
 saveexit := ExitProc;            {Neue Exit-Procedur setzen}
 ExitProc := @NewExit;
END.
```

Völlig namens- und aufrufkompatibel zu den Befehlen der Original-Unit sind die Routinen *ClrScr*, *GotoXY*, *InsLine*, *DelLine*, *KeyPressed*, *ReadKey*,

Delay, *WhereX* und *WhereY*, so daß ich zu diesen Befehlen keinen Kommentar abgeben muß. (Sie sind im Turbo-Pascal-Handbuch beschrieben.)

Die Funktion *Sound* arbeitet ähnlich der Original-Routine, ihr muß jedoch zusätzlich die Dauer des Tons übergeben werden, da die entsprechende BIOS-Routine einen solchen Wert erwartet. Dafür existiert kein Befehl *NoSound*.

Aufrufbeispiel:
Sound(1000,50)
Erzeugt für die Dauer von 50 Millisekunden einen Ton von 1000 Hz.

Die *Textmode*-Funktion schaltet zwischen den drei möglichen Bildschirmmodi des Portfolio um, wobei drei Konstanten zur leichteren Verwendung definiert wurden:

portfolio: Bildschirmmodus 40x25 Zeichen
statisch: Bildschirmmodus 80x25 Zeichen (statisch)
dynamisch: Bildschirmmodus 80x25 Zeichen (dynamisch)

Aufrufbeispiel:
Textmode(portfolio)
Schaltet in den 40x25-Modus um und löscht den Bildschirm.

Im Gegenzug wird durch die Funktion *GetScreenMode* der aktuelle Bildschirmmodus geholt. Falls einer der beiden 80x25-Zeichen-Modi gesetzt wurde, kann der virtuelle Bildschirmausschnitt (40x25 Zeichen) durch die Befehle *MoveScreen* und *ScrollScreen* innerhalb des 80x25-Bildschirms bewegt werden. Der Unterschied zwischen beiden Routinen besteht darin, daß durch *MoveScreen* eine Bewegung zu einem absoluten Punkt vollzogen wird, während durch *ScrollScreen* eine Bewegung relativ zu der aktuellen Position durchgeführt wird. Dabei wurden vier Konstanten *links*, *rechts*, *oben* und *unten* für die Bewegungsrichtung definiert.

1. *MoveScreen(4,5)*
   Die linke obere Ecke des virtuellen Bildschirmausschnitts wird in die Koordinaten 4 (x-Position) und 5 (y-Position) gelegt.

2. *ScrollScreen(4,links)*
   Der virtuelle Bildschirmausschnitt wird um vier Zeichen nach links verschoben.

Über die Funktion *GetScreenPos* kann man die aktuelle Position der linken oberen Ecke des virtuellen Bildschirmausschnitts auslesen: Dabei ist in

dem 16-Bit-Wort die x-Position in den Bits 8 bis 15 und die y-Position in den Bits 0 bis 7 abgelegt.

Die beiden Befehle *StoreBild* und *LoadBild* dienen dazu, einen bestimmten Bildschirmausschnitt in eine Puffervariable abzuspeichern bzw. ihn aus dieser wieder in den Bildschirmspeicher zu kopieren. Die Funktionen sind sehr nützlich, wenn man mit Fenstern arbeiten möchte, nach deren Verschwinden der ursprüngliche Bildschirminhalt ja wieder zu sehen sein muß.

Aufrufbeispiel:
p := StoreBild(0,3,20,10)
Der Bildschirminhalt zwischen den x-Koordinaten 0 und 20 sowie zwischen den y-Koordinaten 3 und 10 wird in die Pointervariable p abgespeichert. Durch den Aufruf *LoadBild(p)* wird dieser Bildschirmbereich wieder rekontruiert.

Der mächtigste Befehl der neuen Unit befähigt Sie, in einem Durchlauf ein typisches Portfolio-Menü, wie es von den internen Applikationen verwendet wird, aufzubauen und auch gleich auszuwerten!. Der *Portmenue*-Routine müssen dabei folgende Parameter übergeben werden:

- ein Menütitel (String)
- die X- und Y-Position der linken, oberen Ecke des Menü-Windows
- die Anzahl der Menüpunkte
- die gewünschte Breite des Menü-Windows in Zeichen
- ein Zeiger auf eine Struktur mit den Menüpunkten

Aufrufbeispiel:

```
......
CONST
menue : array[1..3] of string[20] = ('Menüpunkt1','Menüpunkt2','Menüpunkt3');
VAR
punkt : byte;
......
punkt := PORTMENUE('Demomenü',4,2,3,20,menue);
......
```

Zunächst wird im aufrufenden Programm ein Feld *menue* mit den einzelnen Menüpunkten angelegt (in unserem Fall drei). Die Variable *punkt* enthält den ausgewählten Menüpunkt. Wird durch die [Esc]-Taste abgebrochen, enthält sie den Wert 0 zum Zeichen, daß kein Menüpunkt ausgewählt wurde. Dann wird die Routine aufgerufen: Als Titel erscheint der Text *Demomenü*, die linke obere Ecke des aufgemachten Windows befindet sich an der Position (X=4, Y=2), das Window ist 20 Zeichen breit.

Anschließend wird noch die angelegte Struktur *menue* als Zeiger übergeben. Wenn das Menü aufgebaut ist, können Sie die Menüpunkte – wie von den Applikationen bekannt – auswählen: Entweder geben Sie den ersten Buchstaben des Menüpunktes ein oder bewegen den Cursor mit den Cursor-Tasten auf den gewünschten Menüpunkt, worauf er mit [Return] angewählt werden kann. Einen Abbruch erzwingt man mit [Esc].

Die Unit enthält zusätzlich zu den Pascal-Routinen noch drei Assembler-Prozeduren, die ich Ihnen nun vorstellen möchte. Diese Assembler-Prozeduren sind auf der Begleitdiskette in der Datei *CRTASM.ASM* zusammengefaßt:

```
; NEWINT.ASM :  Neue Interrupt-Routine für Int 21h
;
; Autor: Frank Riemenschneider

IDEAL           ; Ideal-Modus des Turbo Assemblers einschalten
MODEL TPASCAL   ; Speichermodell = TURBO PASCAL wählen


CODESEG
PROC Scroll NEAR
        push dx
        mov ah,06           ;Nach oben scrollen
        mov al,01           ;Um eine Zeile
        mov ch,[ylo]        ;Obere Begrenzung
        mov cl,[xlo]        ;Linke Begrenzung
        mov dh,[yru]        ;Untere Begrenzung
        mov dl,[xru]        ;Rechte Begrenzung
        mov bh,15           ;Farbe
        int 16              ;Scrollen
        pop dx
        ret
ENDP

DATASEG
extrn xru : byte;
extrn xlo : byte;
extrn ylo : byte;
extrn yru : byte;

CODESEG

PROC NewInt FAR
        PUBLIC NewInt

        cmp ah,14           ;Interrupt-Funktion prüfen
        jnz altint          ;Keine Zeichenausgabe, dann alter Interrupt

        push bx
```

```
        push cx
        push si
        push di
        push es
        push dx
        push ds
        push ax
        mov ax,0
        mov ds,ax           ;Datensegment aus Speicherstellen
        mov bx,[0024]       ;$0000:$0018/$0000:$0019 holen
        mov ds,bx

        mov ah,03           ;Cursorposition holen
        mov bh,00
        int 16

        pop ax              ;Zeichencode holen
        push ax

        cmp al,10           ;Linefeed?
        jnz noline
        cmp [yru],dh        ;Untere Begrenzung erreicht?
        jna scrollen        ;Ja, scrollen
        inc dh              ;Zeile erhöhen
        jmp noscroll        ;Sonderzeichen Ende

noline: cmp al,13           ;Return?
        jnz normal
        mov dl,[xlo]        ;Cursor auf linke Begrenzung
        jmp noscroll        ;Sonderzeichen Ende

normal:
        cmp dl,[xru]
        jbe altezeile       ;Rechte Fensterbegrenzung?
        inc dh              ;Ja, Zeile erhöhen
        mov dl,[xlo]        ;Cursor auf linke Begrenzung
altezeile:
        cmp [yru],dh        ;unterer Fensterrand erreicht?
        jnb noscroll        ;Nein, noch nicht!
        dec dh
scrollen:
        call scroll         ;Fenster um eine Zeile nach oben schieben
noscroll:
        mov bh,00
        mov ah,02           ;Cursor positionieren
        int 16

        pop ax
        pop ds
        pop dx
        pop es
        pop di
        pop si
        pop cx
```

```
        pop bx

        cmp al,13           ;Bei RETURN oder
        jz newexit
        cmp al,10           ;Linefeed keinen alten Interrupt!
        jz newexit

altint:
        int 10              ;Alten Interrupt aufrufen
newexit:
        iret

ENDP

; SAVE.ASM :  Speichert Bildschirmausschnitt in Pointer-Variable
;
; Autor: Frank Riemenschneider

PROC Save FAR spseg : word, spofs : word, groesse : word, x1 : word, y1 : word, x2 : word,
                                                                                     y2 : word
   PUBLIC Save
   push ax
   push bx
   push cx
   push si
   push di
   push ds
   push es

   mov ax,[spseg]
   mov ds,ax
   mov di,[spofs]          ;Speicheradresse Pointer laden
   mov ax,[di+2]           ;Segment Speicherbereich holen
   mov es,ax
   mov di,[di]             ;Offset Speicherbereich holen
   mov ax,[groesse]        ;Größe reservierter Bereich

   cld                     ;immer inkrementieren
   stosw                   ;speichern

   mov ax,[y1]
   mov bx,160
   mul bx                  ;Startadresse Video-RAM berechnen:
   mov bx,[x1]
   shl bx,1                ;Y1*160+X1*2
   add ax,bx
   mov si,ax               ;= Offset
   mov ax,45056
   mov ds,ax               ;Segmentadresse Video-RAM

   mov ax,[x1]
   stosw
```

```
    mov ax,[y1]            ;Koordinaten abspeichern
    stosw
    mov ax,[x2]
    sub ax,[x1]            ;(x2-x1+1) Bytes in X-Richtung
    inc ax
    mov bx,ax              ;merken und
    stosw                  ;abspeichern
    mov ax,[y2]
    sub ax,[y1]
    inc ax                 ;(y2-y1+1) Bytes in Y-Richtung
    stosw                  ;abspeichern
    mov cx,ax              ;Als Zähler setzen

Loop1:
    push cx
    push si

    mov cx,bx              ;Anzahl X-Bytes holen
    rep movsw              ;Speicherworte kopieren

    pop ax
    add ax,160             ;Neue Zeile anfangen
    mov si,ax

    pop cx
    loop Loop1             ;Nächste Zeile

    pop es
    pop ds
    pop di
    pop si
    pop cx
    pop bx
    pop ax
    ret

ENDP

; LOAD.ASM :  Lädt Bildschirmdaten aus Pointer-Variablen ein
;
; Autor: Frank Riemenschneider

PROC Load FAR speseg : word, speofs : word
    PUBLIC Load
    push bx
    push cx
    push si
    push di
    push ds
    push es

    mov ax,[speseg]
    mov ds,ax
    mov si,[speofs]        ;Speicheradresse Pointer laden
```

```
    mov ax,[si+2]           ;Segment  Speicherbereich holen
    mov si,[si]             ;Offset Speicherbereich holen
    mov ds,ax

    cld
    lodsw                   ;Größe reservierter Speicherbereich holen
    push ax                 ;und merken
    lodsw                   ;x-Koordinate holen
    mov bx,ax
    lodsw                   ;y-Koordinate holen

    mov cx,160
    mul cx                  ;Startadresse Video-RAM berechnen:
    shl bx,1                ;Y1*160+X1*2
    add ax,bx
    mov di,ax               ;= Offset
    mov ax,45056
    mov es,ax               ;Segmentadresse Video-RAM

    lodsw                   ;Anzahl x-Bytes pro Zeile holen
    mov bx,ax               ;Als Zähler setzen
    lodsw                   ;Anzahl y-Bytes pro Zeile holen
    mov cx,ax

Loop2:
    push cx
    push di

    mov cx,bx               ;Anzahl x-Bytes holen
    rep movsw               ;Speicherworte kopieren

    pop ax
    add ax,160              ;Neue Zeile anfangen
    mov di,ax

    pop cx
    loop Loop2              ;Nächste Zeile

    mov ah,18               ;Bildschirm refreshen
    int 97

    pop ax                  ;Bereichsgröße zurückliefern
    pop es
    pop ds
    pop di
    pop si
    pop cx
    pop bx
    ret

ENDP

END
```

Das größte Problem besteht in der Window-Programmierung. Hierzu muß man wissen, daß Turbo Pascal für die Bildschirmausgaben mit Hilfe der *Write*-Prozedur eine Funktion des Interrupt $21 aufruft, nämlich die Funktion 14. Diese hat die Eigenschaft, daß sie den Text immer bis zum rechten Bildschirmrand ausgibt, dann den Cursor auf die x-Position Null stellt, die Zeile erhöht und dann mit der Textausgabe fortfährt. So sollte es auch sein, wenn da nicht die Windows wären.

Ein Window ist eine rein logische Definition eines Bildschirmbereichs, in dem alle weiteren Ausgaben zu erfolgen haben. Die Zeichenausgaberoutine des Interrupt $21 interessiert das aber natürlich herzlich wenig, was wir definieren, so daß sie fleißig über unsere Window-Grenzen hinwegschreibt. Uns bleiben im Prinzip nur zwei Möglichkeiten: Entweder programmiert man eine völlig neue *Write*-Routine oder man lenkt den Interrupt $21 auf eine eigene Routine um, die in der Lage ist, die Window-Grenzen zu respektieren.

Die erste Variante verbietet sich, wenn man bedenkt, daß mit Hilfe der *Write*-Routine auch Dateien aller Art beschrieben werden können – oder möchten Sie vielleicht das komplette Datei-Handling neu programmieren, so daß es zudem auch noch zu den Prozeduren *Assign*, *Reset* etc. kompatibel bleibt? Ich kann mir dieses ersparen, so daß wir auf die zweite Variante zurückgreifen müssen.

In der Routine *NewInt* wird zunächst geprüft, ob die Funktion 14 des Interrupt $21 aufgerufen wurde. Ist dies nicht der Fall, wird in die alte Routine gesprungen, die auf einen anderen, beim Portfolio unbenutzten Interrupt umgeleitet wurde (INT 10). In der neuen Routine für die Funktion 14 wird nun Zeichen für Zeichen ausgegeben, wobei ständig die Window-Grenzen überprüft werden. Diese sind in den Variablen *Xlo* (x-Koordinate linke Windowbegrenzung), *Ylo* (y-Koordinate obere Windowbegrenzung), *Xru* (x-Koordinate rechte Windowbegrenzung) und *Yru* (y-Koordinate untere Windowbegrenzung) gespeichert. Ist das Zeilenende erreicht, wird der Cursor auf die linke Windowgrenze gesetzt und die nächste Zeile begonnen. Es ist jedoch ein Sonderfall zu beachten: Wird die untere Windowgrenze erreicht, muß das Fenster um eine Zeile nach oben gerollt werden, damit unten Platz für die neue Zeile geschaffen wird (Routine *Scroll*).

Die Prozeduren *Load* und *Save* sind aus Geschwindigkeitsgründen programmiert worden und dienen zum Datentransfer zwischen dem Bildschirminhalt (Video-RAM) und der Puffervariablen. Zwar hätte man dies auch in Pascal programmieren können, die hier verwendeten String-

Befehle des 8088-Prozessors sind dem Pascal-Code jedoch um ein vielfaches überlegen.

Zum Abschluß möchte ich Ihnen noch ein kleines Demo-Programm für die neue Unit vorstellen, die mit der Anweisung

uses portcrt;

in Ihre Programme eingebunden werden kann. Bei der Demonstration der Menüs werden Sie bezüglich Ihres Wissens über die beste deutsche Country-Band, Truck-Stop aus Maschen, befragt. Je nachdem, ob Sie die Fragen richtig beantworten können, wird ein entsprechender Kommentar ausgegeben.

```
{Demoprogramm für Portfolio-CRT-Unit
 written by Truck-Stop-Fan Frank Riemenschneider
           Postfach 730309
           3000 Hannover 71}

{$M 2000,0,10000}
USES dos,Portcrt;

VAR i,j : word;
CONST
menuel : array[1..6] of string = ('Erich Doll','Teddy Ibing','Lucius Reichling',
                                  'Cisco Berndt','Knut Bewersdorff','Uwe Lost');

menue2 : array[1..7] of string = ('Arizona','Keep it Country...','Nicht zu bremsen',
                                  'Lousiana Ladies','Zu Hause','Alles Klar','In Concert');
menue3 : array[1..4] of string = ('Stillhorn','Maschen','Ahrensburg',
                                  'Ottmarschen');
s : string = 'Dies ist eine Demonstration des dynamischen Bildschirmmodus (80*25)!';
BEGIN
  Textmode(statisch);
  writeln;
  writeln;
  writeln('Dieses ist eine Demonstration des statischen',
          ' Bildschirm-Modus (80*25)!');
  FOR i:= 1 to 40 DO BEGIN
    ScrollScreen(1,rechts);
    Delay(400);
  END;
  MoveScreen(0,0);
  GotoXY(0,7);
  writeln('Das Fenster muß nachgeführt werden!');
  FOR i:= 1 to 6 DO BEGIN
    ScrollScreen(1,unten);
    Delay(400);
```

```
END;
MoveScreen(0,0);
ClrScr;
FOR i:= 1 to 10 DO BEGIN
  writeln('Als nächstes folgt eine Window-Demonstration :');
END;
window(10,2,25,6);
ClrScr;
write('Dieser String wurde an der Windowgrenze abgeschnitten!');
Delay(4000);
InsLine(2);
Delay(4000);
DelLine(2);
Delay(4000);
window(0,0,xmax,ymax);
i := portmenue(' Musiker ',10,0,6,20,menue1);
ClrScr;
writeln('Ihr Lieblingsmusiker : ',menue1[i]);
Delay(3000);

Textmode(dynamisch);
writeln;
writeln;
FOR i:= 1 to length(s) DO BEGIN
  write(copy(s,i,1));
  Delay(400);
END;
MoveScreen(0,0);
GotoXY(0,7);
FOR i:= 1 to 15 DO BEGIN
  writeln('Zeile ',i,' wird automatisch nachgeführt!');
  Delay(400);
END;
MoveScreen(0,0);
ClrScr;
FOR i:= 1 to 10 DO BEGIN
  writeln('Als nächstes folgt eine Window-Demonstration :');
END;
window(10,2,25,6);
ClrScr;
write('Dieser String wurde an der Windowgrenze abgeschnitten!');
Delay(4000);
InsLine(2);
Delay(4000);
DelLine(2);
Delay(4000);
window(0,0,xmax,ymax);
i := portmenue(' Platte ',40,0,7,20,menue2);
ClrScr;
writeln('Ihre Lieblingsplatte : ',menue2[i]);
Delay(4000);
```

```
Textmode(portfolio);
writeln;
writeln;
writeln('Im Portfolio-Modus stehen nur 40*25 Zeichen zur Verfügung, so daß',
        ' die Zeile nach 40 Zeichen abgeschnitten wird!');
Delay (4000);
ClrScr;
FOR i:= 1 to 6 DO BEGIN
  writeln('Auch hier können Windows erzeugt werden');
END;
window(2,2,25,6);
ClrScr;
write('Dieser String wurde an der Windowgrenze abgeschnitten!');
Delay(4000);
InsLine(1);
Delay(4000);
DelLine(1);
Delay(4000);
window(0,0,xmax,ymax);
ClrScr;
write('Welcher Ort ist Seevetal 3?');
i := portmenue(' Raten ',3,1,4,15,menue3);
ClrScr;
IF i = 2 THEN BEGIN
   writeln('Wunderbar, Sie sind ein echter Cowboy!');
END
ELSE BEGIN
  writeln('Falsch, Sie werden nie ein Country-Fan!');
END;
END.
```

2.5 Eine GRAPH-Unit für den Portfolio

Als ich meinem Stammhändler COM-DATA in Hannover erzählte, daß man auf dem Portfolio auch Grafikprogrammierung realisieren könne, nickte dieser zwar beifällig, an seinem Gesichtsausdruck konnte man jedoch erkennen, daß er glaubte, ich müsse wohl in die Klapsmühle eingewiesen werden. Nun, irgendwie ist mir das auch verständlich, denn in dem Bedienerhandbuch des Rechners wird nicht ein einziges Wort über den Grafikmodus verloren. Da es sich auch noch um einen relativ kleinen LCD-Bildschirm handelt, kann man nicht unbedingt von der Grafikfähigkeit des Computers ausgehen.

Da die GRAPH-Unit von Turbo Pascal aufgrund des inkompatiblen Grafikmodus nicht läuft, muß man auch hier in den sauren Apfel beißen und eine eigene Unit erstellen. Aus Geschwindigkeitsgründen war es dabei erforderlich, alle elementaren Grafikroutinen in Assembler zu programmieren, wodurch der Aufwand zwar deutlich anstieg, die Ergebnisse dafür aber zufriedenstellen können. Die eigentlichen Grafikbefehle sind wieder namens- und aufrufkompatibel zu den Befehlen der Original-Unit, so daß es in den meisten Fällen reichen wird, die USES-Anweisung auszutauschen. Im einzelnen wurden folgende Grafikroutinen übernommen:

Arc, Bar, Bar3D, Box, Circle, ClearDevice, CloseGraph, DrawPoly, Ellipse, FillEllipse, FillPoly, FloodFill, GetColor, GetPixel, InitGraph, Line, LineRel, LineTo, MoveRel, MoveTo, OutTextXY, PieSlice, PutPixel, Rectangle, Sector, SetColor, SetFillStyle, SetTextStyle, TextPixel.

Bei den folgenden Befehlen sind Abweichungen zu denen der Original-Unit notwendig geworden:

Bei *OutTextXY* können nur ASCII-Codes bis einschließlich 127 ausgegeben werden. Da kein ROM-Zeichensatz existiert, mußten diese Codes mühsam per Hand zusammengebastelt werden. Da ab Code 128 fast nur noch Grafikzeichen kommen, lohnt sich meiner Meinung nach der Zeitaufwand nicht.

Der Befehl *SetTextStyle* besitzt eine etwas andere Aufrufsyntax: Der erste Original-Parameter *(font)* existiert zwar ebenfalls, hat jedoch keine Wirkung auf den Zeichensatz. Für den Parameter *direction* sind Werte von 1 bis 12 erlaubt, wobei das Aussehen der Zeichen wie folgt beeinflußt wird:

1: Zeichen nicht gekippt, keine Kursivschrift
2: Zeichen nicht gekippt, Kursivschrift nach links
3: Zeichen nicht gekippt, Kursivschrift nach rechts
4: Zeichen um 180° gekippt, keine Kursivschrift
5: Zeichen um 180° gekippt, Kursivschrift nach links
6: Zeichen um 180° gekippt, Kursivschrift nach rechts
7: Zeichen um 90° gekippt, keine Kursivschrift
8: Zeichen um 90° gekippt, Kursivschrift nach links
9: Zeichen um 90° gekippt, Kursivschrift nach rechts
10: Zeichen um 270° gekippt, keine Kursivschrift
11: Zeichen um 270° gekippt, Kursivschrift nach links
12: Zeichen um 270° gekippt, Kursivschrift nach rechts

Statt eines weiteren Parameters bei der Original-Unit, der dort die Zeichengröße gemeinsam für x- und y-Richtung festlegt, folgen bei der Portfolio-Unit noch zwei, die die Zeichengröße in x- und y-Richtung getrennt festlegen.

Aufrufbeispiel: *SetTextStyle(0,3,2,1)*
Der folgende Text wird rechtskursiv ausgegeben, wobei in x-Richtung eine Vergrößerung um Faktor 2 und in y-Richtung keine Vergrößerung vorgenommen wird.

Aus Kompatibilitätsgründen wurde die Routine *InitGraph* original übernommen. Das bedeutet, daß ein Pfadname für einen Grafiktreiber (im Original .BGI) angegeben werden muß. Da die Portfolio-Unit aber gar keinen Treiber benötigt, können Sie eingeben, was Sie wollen.

Die Tabellen *adrtab* und *costab* werden von der Textpixel- bzw. Ellipsen-Assembler-Routine benötigt. Bedingt durch das Speichermodell akzeptiert der Turbo Assembler keine initialisierten Variablen im Assembler-Modul, so daß diese im Pascal-Programm zur Verfügung gestellt werden müssen.

```
UNIT portgraf;
{written by Frank Riemenschneider
            Postfach 730309
            3000 Hannover 71}

{$M 5000,0,0}
{$L a:portgraf.obj}

INTERFACE

PROCEDURE SetColor (Color : WORD);
PROCEDURE CloseGraph;
PROCEDURE InitGraph(VAR driver : integer; VAR mode : integer; path : string);
PROCEDURE InitGraphic;
PROCEDURE PutPixel (x, y : Integer; Color : word);
PROCEDURE Line (x1, y1, x2, y2 : Integer);
PROCEDURE Rectangle (x1, y1, x2, y2 : Integer);
PROCEDURE Bar (x1, y1, x2, y2 : Integer);
PROCEDURE Circle (xm, ym : Integer; Radius : WORD);
PROCEDURE Ellipse (xm, ym : Integer; AnfWinkel, EndWinkel, XRadius, YRadius : WORD);
PROCEDURE Arc (xm, ym : Integer; AnfWinkel, EndWinkel, Radius : WORD);
PROCEDURE ClearDevice;
PROCEDURE Plot (x, y : WORD);
PROCEDURE Box (x1, y1, x2, y2 : WORD);
PROCEDURE Curve (xm, ym, xr, yr, AnfWinkel, EndWinkel : WORD);
PROCEDURE FloodFill (x,y : Integer; border : word);
PROCEDURE FillEllipse (xm, ym : Integer; XRadius, YRadius : word);
PROCEDURE SetFillStyle (muster,color : word);
```

```
FUNCTION GetColor : word;
FUNCTION GetPixel (x, y : Integer) : word;
FUNCTION TestPixel (x,y : word) : word;
PROCEDURE Slice (xm, ym, xr, yr, AnfWinkel, EndWinkel : WORD);
PROCEDURE Sector (xm,ym: Integer; AnfWinkel, EndWinkel, XRadius, YRadius : word);
PROCEDURE PieSlice (xm,ym: Integer; AnfWinkel, EndWinkel, Radius: word);
PROCEDURE Text (x, y, TextSeg, TextOfs : WORD);
PROCEDURE OutTextXY(x,y : Integer; TextString : STRING);
PROCEDURE SetTextStyle (font,direction,Sizex,Sizey : word);
PROCEDURE Bar3D(x1,y1,x2,y2:integer; depth:word; top:boolean);
PROCEDURE DrawPoly(NumPoints : word; VAR PolyPoints);
PROCEDURE FillPoly(NumPoints : word; VAR PolyPoints);
PROCEDURE MoveRel(dx,dy : integer);
PROCEDURE MoveTo(x,y : integer);
PROCEDURE LineRel(dx,dy : integer);
PROCEDURE LineTo(x,y : integer);
PROCEDURE Fill(x,y: integer; border : word);

TYPE
  PointType = record
          x,y : word;
  END;

CONST

        { Tabelle der Cosinus-Werte }
        costab : ARRAY [0..395] OF BYTE =
        ( 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
            19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
            35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
            51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,
            67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
            83, 84, 85, 86, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
            98, 99, 100, 101, 102, 102, 103, 104, 105, 106, 107, 108, 109,
            110, 111, 112, 113, 114, 114, 115, 116, 117, 118, 119, 120, 121,
            122, 122, 123, 124, 125, 126, 127, 128, 129, 130, 130, 131, 132,
            133, 134, 135, 136, 136, 137, 138, 139, 140, 141, 142, 142, 143,
            144, 145, 146, 147, 147, 148, 149, 150, 151, 152, 152, 153, 154,
            155, 156, 156, 157, 158, 159, 160, 160, 161, 162, 163, 163, 164,
            165, 166, 167, 167, 168, 169, 170, 170, 171, 172, 173, 173, 174,
            175, 176, 176, 177, 178, 179, 179, 180, 181, 181, 182, 183, 184,
            184, 185, 186, 186, 187, 188, 188, 189, 190, 190, 191, 192, 192,
            193, 194, 194, 195, 196, 196, 197, 198, 198, 199, 200, 200, 201,
            201, 202, 203, 203, 204, 204, 205, 206, 206, 207, 207, 208, 209,
            209, 210, 210, 211, 212, 212, 213, 213, 214, 214, 215, 215, 216,
            216, 217, 218, 218, 219, 219, 220, 220, 221, 221, 222, 222, 223,
            223, 224, 224, 225, 225, 226, 226, 227, 227, 227, 228, 228, 229,
            229, 230, 230, 231, 231, 231, 232, 232, 233, 233, 234, 234, 234,
            235, 235, 235, 236, 236, 237, 237, 237, 238, 238, 238, 239, 239,
            240, 240, 240, 241, 241, 241, 242, 242, 242, 243, 243, 243, 243,
            244, 244, 244, 245, 245, 245, 245, 246, 246, 246, 247, 247, 247,
            247, 248, 248, 248, 248, 248, 249, 249, 249, 249, 250, 250, 250,
            250, 250, 251, 251, 251, 251, 251, 251, 252, 252, 252, 252, 252,
            252, 253, 253, 253, 253, 253, 253, 253, 253, 254, 254, 254, 254,
```

```
          254, 254, 254, 254, 254, 254, 254, 254, 255, 255, 255, 255, 255,
          255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255 );

{Bit-Daten des Zeichensatzes}

    daten : array[0..767] of byte =
{ASCII 0}         (0,0,0,0,0,0,
{ASCII 1}          62,85,81,85,62,0,
{ASCII 2}          62,107,111,107,62,0,
{ASCII 3}          30,62,124,62,30,0,
{ASCII 4}          8,28,62,28,8,0,
{ASCII 5}          28,95,103,95,28,0,
{ASCII 6}          28,94,127,94,28,0,
{ASCII 7}          0,0,24,24,0,0,
{ASCII 8}          255,255,231,231,255,255,
{ASCII 9}          0,24,36,36,24,0,
{ASCII 10}         255,231,219,219,231,255,
{ASCII 11}         48,72,77,75,55,0,
{ASCII 12}         6,41,121,41,6,0,
{ASCII 13}         96,96,63,5,7,0,
{ASCII 14}         96,127,5,53,63,0,
{ASCII 15}         42,28,119,28,42,0,
{ASCII 16}         127,62,28,8,8,0,
{ASCII 17}         8,8,28,62,127,0,
{ASCII 18}         20,54,127,54,20,0,
{ASCII 19}         0,95,0,95,0,0,
{ASCII 20}         6,9,127,1,127,0,
{ASCII 21}         0,74,85,85,41,0,
{ASCII 22}         112,112,112,112,112,0,
{ASCII 23}         84,118,127,118,84,0,
{ASCII 24}         4,6,127,6,4,0,
{ASCII 25}         16,48,127,48,16,0,
{ASCII 26}         8,8,42,28,8,0,
{ASCII 27}         8,28,42,8,8,0,
{ASCII 28}         60,32,32,32,0,0,
{ASCII 29}         8,28,8,28,8,0,
{ASCII 30}         32,56,62,56,32,0,
{ASCII 31}         2,14,62,14,2,0,
{ASCII 32}         0,0,0,0,0,0,
{ASCII 33}         0,0,95,0,0,0,
{ASCII 34}         0,3,0,3,0,0,
{ASCII 35}         20,127,20,127,20,0,
{ASCII 36}         36,42,107,42,18,0,
{ASCII 37}         35,19,8,100,98,0,
{ASCII 38}         54,73,85,34,80,0,
{ASCII 39}         0,0,5,3,0,0,
{ASCII 40}         0,28,34,65,0,0,
{ASCII 41}         0,65,34,28,0,0,
{ASCII 42}         20,8,62,8,20,0,
{ASCII 43}         8,8,62,8,8,0,
{ASCII 44}         0,0,80,48,0,0,
{ASCII 45}         8,8,8,8,8,0,
{ASCII 46}         0,0,96,96,0,0,
{ASCII 47}         32,16,8,4,2,0,
```

```
{ASCII 48}          62,81,73,69,62,0,
{ASCII 49}          0,66,127,64,0,0,
{ASCII 50}          66,97,81,73,70,0,
{ASCII 51}          33,65,69,75,49,0,
{ASCII 52}          24,20,18,127,16,0,
{ASCII 53}          39,69,69,69,57,0,
{ASCII 54}          60,74,73,73,48,0,
{ASCII 55}          1,1,121,5,3,0,
{ASCII 56}          54,73,73,73,54,0,
{ASCII 57}          6,73,73,41,30,0,
{ASCII 58}          0,0,54,54,0,0,
{ASCII 59}          0,0,86,54,0,0,
{ASCII 60}          0,8,20,34,65,0,
{ASCII 61}          20,20,20,20,20,0,
{ASCII 62}          65,34,20,8,0,0,
{ASCII 63}          2,1,81,9,6,0,
{ASCII 64}          62,65,73,85,14,0,
{ASCII 65}          126,17,17,17,126,0,
{ASCII 66}          127,74,74,74,54,0,
{ASCII 67}          62,65,65,65,34,0,
{ASCII 68}          127,65,65,34,28,0,
{ASCII 69}          127,73,73,73,65,0,
{ASCII 70}          127,9,9,9,1,0,
{ASCII 71}          62,65,81,81,114,0,
{ASCII 72}          127,8,8,8,127,0,
{ASCII 73}          0,65,127,65,0,0,
{ASCII 74}          32,64,65,63,1,0,
{ASCII 75}          127,8,20,34,65,0,
{ASCII 76}          127,64,64,64,64,0,
{ASCII 77}          127,2,12,2,127,0,
{ASCII 78}          127,4,8,16,127,0,
{ASCII 79}          62,65,65,65,62,0,
{ASCII 80}          127,9,9,9,6,0,
{ASCII 81}          62,65,81,33,94,0,
{ASCII 82}          127,9,25,41,70,0,
{ASCII 83}          38,73,73,73,50,0,
{ASCII 84}          1,1,127,1,1,0,
{ASCII 85}          63,64,64,64,63,0,
{ASCII 86}          31,32,64,32,31,0,
{ASCII 87}          127,32,24,32,127,0,
{ASCII 88}          99,20,8,20,99,0,
{ASCII 89}          7,8,120,8,7,0,
{ASCII 90}          97,81,73,69,67,0,
{ASCII 91}          0,127,65,65,0,0,
{ASCII 92}          2,4,8,16,32,0,
{ASCII 93}          0,65,65,127,0,0,
{ASCII 94}          4,2,1,2,4,0,
{ASCII 95}          128,128,128,128,128,128,
{ASCII 96}          0,3,5,0,0,0,
{ASCII 97}          32,84,84,84,120,0,
{ASCII 98}          127,72,68,68,56,0,
{ASCII 99}          56,68,68,68,32,0,
{ASCII 100}         56,68,68,72,127,0,
{ASCII 101}         56,84,84,84,88,0,
```

```
{ASCII 102}         8,126,9,9,2,0,
{ASCII 103}         8,84,84,84,60,0,
{ASCII 104}         127,8,4,4,120,0,
{ASCII 105}         0,68,125,64,0,0,
{ASCII 106}         32,64,68,61,0,0,
{ASCII 107}         127,32,16,40,68,0,
{ASCII 108}         0,65,127,64,0,0,
{ASCII 109}         124,4,24,4,120,0,
{ASCII 110}         124,8,4,4,120,0,
{ASCII 111}         56,68,68,68,56,0,
{ASCII 112}         124,20,20,20,8,0,
{ASCII 113}         8,20,20,20,124,0,
{ASCII 114}         124,8,4,4,8,0,
{ASCII 115}         72,84,84,84,36,0,
{ASCII 116}         4,63,68,68,32,0,
{ASCII 117}         60,64,64,32,120,0,
{ASCII 118}         28,32,64,32,28,0,
{ASCII 119}         60,64,48,64,60,0,
{ASCII 120}         68,40,16,40,68,0,
{ASCII 121}         76,80,80,80,60,0,
{ASCII 122}         68,100,84,76,68,0,
{ASCII 123}         0,8,62,65,65,0,
{ASCII 124}         0,0,119,0,0,0,
{ASCII 125}         65,65,62,8,0,0,
{ASCII 126}         2,1,3,2,1,0,
{ASCII 127}         96,80,72,80,96,0);
{Tabelle mit Zeilenanfängen des Video-RAM}

        adrtab : ARRAY [0..63] OF WORD =
(0, 30, 60, 90, 120, 150, 180, 210, 240, 270, 300, 330, 360, 390, 420,
450, 480, 510, 540, 570, 600, 630, 660, 690, 720, 750, 780, 810,840,
870, 900, 930, 960, 990, 1020, 1050, 1080, 1110, 1140, 1170, 1200, 1230,
1260, 1290, 1320, 1350, 1380, 1410, 1440, 1470, 1500, 1530, 1560, 1590, 1620, 1650,1680,
1710, 1740, 1770, 1800, 1830, 1860, 1890);

VAR
i, Farbe,fillmuster,fillcolor,ArcStartX, ArcStartY, ArcEndX, ArcEndY : WORD;
zu, zl, su, sl, zch, spt, xver, yver : integer;

gcursorx, gcursory : integer;

IMPLEMENTATION

{$F+}              { Die Assembler-Routinen müssen als FAR-Routinen
                     eingebunden werden }
PROCEDURE InitGraphic; EXTERNAL;
PROCEDURE CloseGraph; EXTERNAL;
PROCEDURE Plot; EXTERNAL;
PROCEDURE Line; EXTERNAL;
PROCEDURE Box; EXTERNAL;
PROCEDURE Bar; EXTERNAL;
PROCEDURE Curve; EXTERNAL;
PROCEDURE Slice; EXTERNAL;
PROCEDURE Text; EXTERNAL;
```

```
PROCEDURE Fill; EXTERNAL;
FUNCTION TestPixel; EXTERNAL;
{$F-}

{Initialisiert Grafikmodus und schreibt Zeichensatz ins Video-RAM}

PROCEDURE InitGraph(VAR driver : integer; VAR mode : integer; path : string);
VAR i : word;
BEGIN
 InitGraphic;
 FOR i:= 0 to 767 DO BEGIN
    Mem[$B000:$07D0+i] := daten[i];
 END;
END;

{Setzt einen Pixel in der angegebenen Farbe}

PROCEDURE PutPixel (x, y : Integer; Color : word);
BEGIN
     Farbe := Color;
     Plot(x, y);
END;

{Testet einen Pixel auf seine Farbe}

FUNCTION GetPixel (x, y : Integer) : WORD;
BEGIN
 GetPixel := TestPixel(x,y);
END;

{Setzt die Zeichenfarbe für die weiteren Grafikbefehle,
 erlaubte Werte sind 0 und 1}

PROCEDURE SetColor (Color : word);
BEGIN
     Farbe := Color;
END;

{Zeichnet nicht gefülltes Rechteck}

PROCEDURE Rectangle (x1, y1, x2, y2 : Integer);
BEGIN
     Box (x1, y1, x2, y2);
END;

{Wandelt Winkel in Format der Original TP-Graph-Unit um}

PROCEDURE winkel (VAR AnfWinkel:word; VAR EndWinkel : word);
VAR
aw,ew : integer;
```

```
BEGIN
  EW := 90-AnfWinkel;
  AW := 90-EndWinkel;
  IF AW < 0 THEN AW := 360+AW;
  IF EW < 0 THEN EW := 360+EW;
  AnfWinkel := AW;
  EndWinkel := EW;
END;

{Zeichnet kompletten Kreis}

PROCEDURE Circle (xm, ym : Integer; Radius : WORD);
BEGIN
     Curve (xm, ym, Radius, Radius, 0, 360);
END;

{Füllt beliebige Fläche mit Farbe aus}

procedure FloodFill(x,y : integer; border : word);
BEGIN
  Fill(x,y,border);
END;

{Setzt Füllmodus}

PROCEDURE SetFillStyle (muster,color : word);
BEGIN
  fillmuster := muster;
  fillcolor :=  color;
END;

{Zeichnet Ellipsen(ausschnitt)}

PROCEDURE Ellipse (xm, ym : Integer; AnfWinkel, EndWinkel, XRadius, YRadius : word);
BEGIN
  winkel(AnfWinkel,EndWinkel);
  Curve (xm, ym, XRadius, YRadius, AnfWinkel, EndWinkel);
END;

{Zeichnet ausgefüllte Ellipse}

PROCEDURE FillEllipse (xm, ym : Integer; XRadius, YRadius : word);
BEGIN
     Curve (xm, ym, XRadius, YRadius, 0, 360);
     IF fillmuster <> 0 THEN Fill(xm,ym,farbe);
END;

{Zeichnet Kreisbogenausschnitt}

PROCEDURE Arc (xm, ym : Integer; AnfWinkel, EndWinkel, Radius : WORD);
BEGIN
     winkel(AnfWinkel,EndWinkel);
     Curve (xm, ym, Radius, Radius, AnfWinkel, EndWinkel);
END;
```

```
{Zeichnet ausgefülltes Tortenstück}

PROCEDURE Sector (xm, ym : Integer; AnfWinkel, EndWinkel, XRadius, YRadius : WORD);
VAR halbierende : real;
BEGIN
     halbierende := ((Endwinkel+Anfwinkel)/2)*PI/180;
     winkel(AnfWinkel,EndWinkel);
     Slice (xm, ym, XRadius, YRadius, AnfWinkel, EndWinkel);
     IF fillmuster <> 0 THEN Fill(xm+Round(Xradius*cos(halbierENDe)/2),
             ym-Round(Yradius*sin(halbierende)/2),farbe);
END;

{Zeichnet nicht ausgefülltes Tortenstück}

PROCEDURE PieSlice (xm, ym : Integer; AnfWinkel, EndWinkel, Radius : WORD);
BEGIN
     Sector(xm, ym, AnfWinkel, EndWinkel, Radius, Radius);
END;

{Löscht Grafikbildschirm}

PROCEDURE ClearDevice;
BEGIN
     InitGraphic;
END;

{Holt aktuelle Zeichenfarbe}

FUNCTION GetColor : word;
BEGIN
     GetColor := Farbe;
END;

{Zeichnet dreidimensionale Säule}

PROCEDURE Bar3D(x1,y1,x2,y2:integer; depth:word; top:boolean);
BEGIN
   bar(x1,y1,x2,y2);
   line(x2,y2,x2+depth,y2-depth);
   line(x2+depth,y2-depth,x2+depth,y1-depth);
   IF top THEN BEGIN
     line(x2,y1,x2+depth,y1-depth);
     line(x1,y1,x1+depth,y1-depth);
     line(x1+depth,y1-depth,x2+depth,y1-depth);
   END;
END;

{Zeichnet Polygon}

PROCEDURE DrawPoly(NumPoints : word; VAR PolyPoints);
TYPE wordes = array[1..20] of PointType;
VAR i : byte;
```

```
BEGIN
 FOR i:= 2 to NumPoints DO BEGIN
   line(wordes(PolyPoints)[i-1].x,wordes(PolyPoints)[i-1].y,wordes(PolyPoints)[i]
                                                    .x,wordes(PolyPoints)[i].y);
 END;
END;

{Zeichnet ausgefülltes Polygon}

PROCEDURE FillPoly(NumPoints : word; VAR PolyPoints);
TYPE wordes = array[1..20] of PointTYPE;
VAR x,y : integer;
BEGIN
  DrawPoly(NumPoints,PolyPoints);
  IF ((wordes(PolyPoints)[1].x <> wordes(PolyPoints)[NUmPoints].x) or
      (wordes(PolyPoints)[1].y <> wordes(PolyPoints)[NUmPoints].y)) THEN BEGIN

line(wordes(PolyPoints)[1].x,wordes(PolyPoints)[1].y,wordes(PolyPoints)[NumPoints].x,wordes
(PolyPoints)[NumPoints].y);
        inc(NumPoints);
        wordes(PolyPoints)[NumPoints].x := wordes(PolyPoints)[1].x;
        wordes(PolyPoints)[NumPoints].y := wordes(PolyPoints)[1].y;
  END;
  x := wordes(PolyPoints)[1].x + Round((wordes(POlyPoints)[NumPoints-2].x-wordes
                                                      (PolyPoints)[1].x)/2);
  y := wordes(PolyPoints)[1].y + Round((wordes(POlyPoints)[NumPoints-2].y-wordes
                                                      (PolyPoints)[1].y)/2);
  x := x + Round((wordes(PolyPoints)[NumPoints-1].x-x)/2);
  y := y + Round((wordes(PolyPoints)[NumPoints-1].y-y)/2);
  IF ((fillmuster <>0) and (NumPoints>2)) THEN Fill(x,y,farbe);
END;

{Bewegt Grafikcursor relativ zur aktuellen Position}

PROCEDURE MoveRel(dx,dy : integer);
BEGIN
  gcursorx := gcursorx + dx;
  gcursory := gcursory + dy;
END;

{Bewegt Grafikcursor zu einer absoluten Position}

PROCEDURE MoveTo(x,y : integer);
BEGIN
  gcursorx := x;
  gcursory := y;
END;

{Zeichnet Linie zu einem relativen Punkt vom Grafikcursor ausgehend}

PROCEDURE LineRel(dx,dy : integer);
BEGIN
  Line(gcursorx,gcursory,gcursorx+dx,gcursory+dy);
END;
```

```
{Zeichnet Linie zu einem absoluten Punkt von Grafikcursor ausgehend}

PROCEDURE LineTo(x,y : integer);
BEGIN
  Line(gcursorx,gcursory,x,y);
END;

{Schreibt Text in Grafikbildschirm}

PROCEDURE OutTextXY(x,y : Integer; TextString : STRING);
BEGIN
    Text(x, y, Seg(TextString), Ofs(TextString));
END;

{Setzt Text-Ausgabemodus}

PROCEDURE SetTextStyle (font,direction,Sizex,Sizey : word);
CONST
zeiun : array[1..12] of integer = (1,1,1,-1,-1,-1,0,1,-1,0,-1,1);
zeili : array[1..12] of integer = (0,-1,1,0,-1,1,-1,-1,-1,1,1,1);
spaun : array[1..12] of integer = (0,0,0,0,0,0,1,1,1,-1,-1,-1);
spali : array[1..12] of integer = (1,1,1,-1,-1,-1,0,0,0,0,0,0);
zeiof : array[1..12] of integer = (6,6,6,-6,-6,-6,0,0,0,0,0,0);
spaof : array[1..12] of integer = (0,0,0,0,0,0,6,6,6,-6,-6,-6);
BEGIN
 IF ((direction >=1) and (direction <=12)) THEN BEGIN
  xver := Sizex;
  yver := Sizey;
  zu := zeiun[direction];
  zl := zeili[direction];
  su := spaun[direction];
  sl := spali[direction];
  zch := zeiof[direction]*xver;
  spt := spaof[direction]*xver;
 END;
END;

{Initialisierungsroutine}

BEGIN
 zu := 1;          {Textausrichtung normal}
 zl := 0;
 su := 0;
 sl := 1;
 xver := 1;        {Vergößerungsfaktor 1}
 yver := 1;
 zch := 6;
 spt := 0;
 farbe := 1;
 fillmuster := 1;
```

```
fillcolor := 1;
gcursorx := 0;
gcursory := 0;     {Grafikcursor}
END.
```

Zum Pascal-Teil möchte ich nichts mehr sagen, da die einzelnen Unterprogramme relativ primitiv aufgebaut sind und Sie weitere Informationen zu den Grafikbefehlen aus dem Handbuch zu Turbo Pascal entnehmen können. Wesentlich interessanter scheint es mir, auf die Arbeitsweise der Assembler-Routinen einzugehen, die trotz der geringen Taktfrequenz des Portfolio-Prozessors ein angenehmes Arbeiten zulassen.

2.5.1 Grafikmodus initialisieren

Starten wir mit der einfachsten Aufgabe, dem Ein- und Ausschalten des Grafikmodus: Wie aus Kapitel 1 bekannt ist, muß man für die Grafik den (beim PC unbekannten) Videomodus 10 setzen, das Ausschalten der Grafik wird durch Setzen des Textmodus (Videomodus 7) erreicht.

```
; INIT.ASM:      Initialisiert Grafik-/Textschirm
;
; Autor: Frank Riemenschneider
;

CODESEG

PROC InitGraphic FAR

        PUBLIC InitGraphic
        mov ah, 00
        mov al, 0Ah
        int 10h
        ret

ENDP

PROC CloseGraph FAR
        PUBLIC CloseGraph

        mov ah, 00
        mov al, 07
        int 10h
        ret

ENDP
```

2.5.2 Grafikpunkt setzen

Normalerweise gehöre ich zu den Leuten, die von der Benutzung von Betriebssystemroutinen zum Setzen eines Grafikpunktes abraten und statt dessen empfehlen, direkt das Video-RAM zu beschreiben. Man muß sich nur einmal klar machen, wie oft die Plot-Routine von allen anderen Grafikbefehlen wie z.B. *Ellipse*, *Line* etc. aufgerufen wird, um festzustellen, daß hier minimale Geschwindigkeitsvorteile im Endeffekt deutliche Zeitgewinne verursachen können. Beim Portfolio sieht die Sache jedoch anders aus: Zum einen ist der Bildschirm relativ klein, so daß deutlich weniger Grafikpunkte pro Grafikbefehl gezeichnet werden müssen als z.B. bei einer VGA-Karte mit einer Auflösung von 640*400 Punkten. Der Hauptgrund liegt jedoch in dem notwendigen Bildschirm-Refresh: Würde man einfach nur das Video-RAM beschreiben, sähe man überhaupt nichts. Der Refresh mittels Interrupt $61 dauert jedoch solange, daß kein Geschwindigkeitsvorteil gegenüber der Betriebssystemroutine mehr feststellbar wäre. Die folgende Routine prüft zunächst, ob sich die Koordinaten im zulässigen Rahmen bewegen und ruft dann den Interrupt $10 auf.

```
; ASMPLOT.ASM: Setzt einen Punkt im Grafik-Bildschirm
;
; Autor: Frank Riemenschneider
;
; Eingabe:      CX = x-Koordinate (0-239)
;               DX = y-Koordinate (0-63)
;
; Ausgabe:      keine


DATASEG
extrn Farbe : word
extrn gcursorx : word
extrn gcursory : word

CODESEG

PROC AsmPlot
        push ax
        push bx                 ; Register auf
        push cx                 ; Stack retten
        push dx
        push si
        push di
```

```
        mov [gcursorx],ax
        mov [gcursory],bx

        mov cx,ax
        mov dx,bx
        mov ah, 12
        mov al, [BYTE farbe]
        int 16

        pop di                  ; Alle Register wieder vom
        pop si                  ; Stack herunterholen
        pop dx
        pop cx
        pop bx
        pop ax
        ret                     ; Back home

ENDP


; PLOT.ASM:     Regelt die Parameterübernahme von Turbo Pascal aus für
;               die ASMPLOT-Routine in Turbo Assembler
;
; Autor: Frank Riemenschneider
;
; Eingabe:      Parameter der PLOT-Funktion auf dem Stack
;               Plot( X, Y:WORD );
; Ausgabe:      keine

CODESEG

PROC Plot FAR Xkoor:WORD, Ykoor:WORD
        PUBLIC Plot

        mov ax, [Xkoor]         ; x- und y-Koordinaten in die
        mov bx, [Ykoor]         ; entsprechenden Register bringen
        cmp ax, 0               ; x-Koordinate < 0?
        jl PlotExit             ; Ja -> Rücksprung
        cmp ax, 239             ; x-Koordinate > 239
        jg PlotExit             ; Ja -> Rücksprung
        cmp bx, 0               ; y-Koordinate < 0?
        jl PlotExit             ; Ja -> Rücksprung
        cmp bx, 63              ; y-Koordinate > 63?
        jg PlotExit             ; Ja -> Rücksprung
        call AsmPlot            ; Plot-Routine aufrufen
PlotExit:
        ret                     ; Back home

ENDP
```

2.5.3 Grafikpunkt testen

Die *Test*-Routine scheint auf den ersten Blick ebenfalls dazu angetan, den BIOS-Interrupt zu benutzen. Doch Achtung! Eine Flächenfüllroutine, wie wir sie unten kennenlernen werden, muß diese Routine permanent aufrufen, um festzustellen, ob ein Grafikpunkt gesetzt ist oder nicht. Da eine Flächenfüllroutine sowieso mit zu den zeitkritischsten Aufgaben der Grafikprogrammierung gehört, ist eine Geschwindigkeitsmaximierung auch bei allen Unter-Routinen erforderlich. Deshalb werden wir nicht den BIOS-Interrupt benutzen, sondern das Video-RAM direkt auslesen, um auch einen noch so kleinen Zeitgewinn zu ermöglichen. Da im Gegensatz zum Setzen eines Punktes kein Bildschirm-Refresh erforderlich ist, kann man den Vorteil des Verfahrens zu 100% ausnutzen. In Kapitel 1 ist der Aufbau des Portfolio-Video-RAM beschrieben, so daß das folgende Listing leicht zu verstehen ist. Die Routine gibt den Farbwert zurück, der beim Setzen des Punktes angegeben wurde.

```
; CALCXY.ASM:   Berechnet Segment und Offset des Bytes im Video-RAM, sowie
;               die Bitmaske für das passende Bit bei gegebener x- und
;               y-Koordinate
;
; Autor: Frank Riemenschneider
;
; Eingabe:      AX = x-Koordinate (0-239)
;               BX = y-Koordinate (0-63)
; Ausgabe:      ES = Segmentadresse des Bytes im Video-RAM
;               DI = Offsetadresse des Bytes im Video-RAM
;               DL = Bitmaske des Bytes im Video-RAM

DATASEG

extrn   adrtab : DWORD

CODESEG

PROC CalcXY

        push cx
        push si
        mov cx, ax              ; Koordinaten nochmal für spätere
                                ; Verwendung in der Berechnung retten
        mov dx, 0b000h          ; Offset EGA-Karte laden
        lea si, adrtab
                                ; Adresse der Zeilenanfänge der EGA-Karte laden
        mov es, dx              ; Segmentadresse laden
```

```
; Offsetadresse berechnen
        shl bx, 1               ; Tabelle enthält Wort-Werte, also
                                ; y-Koordinate verdoppeln
        mov dx, [si+bx]         ; Adresse des Zeilenanfangs laden
        shr ax, 1
        shr ax, 1
        shr ax, 1               ; x/8
        add dx, ax              ; Spaltenoffset addieren
        mov di, dx              ; DI mit Offset laden

; Berechnung der Bitmaske
        and cl, 7               ; x MOD 8
        mov dl, 80h             ; Maske für Bit 0
        shr dl, cl              ; Maske für gewünschtes Bit durch Rotation
                                ; der ursprünglichen Maske erzeugen
        pop si
        pop cx
        ret                     ; Back home

ENDP CalcXY

; ASMTEST.ASM: Testet einen Punkt im Grafik-Bildschirm
;
; Autor: Frank Riemenschneider
;
; Eingabe:      AX = x-Koordinate (0-239)
;               BX = y-Koordinate (0-63)
; Ausgabe:      DX = Farbe des Punktes

DATASEG

extrn farbe : word

CODESEG

PROC AsmTest

        push cx                 ; Stack kommen
        push si
        push di
        mov dl, [BYTE farbe]    ; Default: Punkt gesetzt
        call CalcXY             ; Offset- und Segmentadresse sowie die
                                ; Bitmaske aus den x- und y-Koordinaten
                                ; berechnen
        mov ch, [es:di]         ; Byte aus dem Video-RAM lesen
        and dl, ch              ; Bit testen
        jz TestExit1            ; Ergebnis 0 => schwarz
        mov dl, [BYTE farbe]    ; Punkt ist weiß

TestExit1:
        xor dh, dh              ; Hi-Byte löschen
        pop di                  ; Alle Register wieder von den Bäumen,
        pop si                  ; äh, vom Stack herunterholen
```

```
        pop cx
        ret                     ; Back home

ENDP


; TEST.ASM:     Regelt die Parameterübernahme von Turbo Pascal aus für
;               die ASMTEST-Routine in Turbo Assembler
;
; Autor: Frank Riemenschneider
;
; Eingabe:      Parameter der TESTPIXEL-Funktion auf dem Stack
;               TestPixel( X, Y : WORD );
; Ausgabe:      Farbe des Punktes

CODESEG

PROC TestPixel FAR Xkoord:WORD, Ykoord:WORD
        PUBLIC TestPixel

        mov ax, [Xkoord]        ; x- und y-Koordinaten in die
        mov bx, [Ykoord]        ; entsprechenden Register bringen
        cmp ax, 0               ; x-Koordinate < 0?
        jl TestExit             ; Ja -> Rücksprung
        cmp ax, 239             ; x-Koordinate > 239?
        jg TestExit             ; Ja -> Rücksprung
        cmp bx, 0               ; y-Koordinate < 0?
        jl TestExit             ; Ja -> Rücksprung
        cmp bx, 63              ; y-Koordinate > 63?
        jg TestExit             ; Ja -> Rücksprung
        call AsmTest            ; Test-Routine aufrufen
        xor dh, dh
        mov ax, dx              ; Farbe zurückgeben
TestExit:
        ret                     ; Back home

ENDP
```

2.5.4 Linie ziehen

Nach den relativ einfachen Grundfunktionen (Punkt setzen, Punkt testen) kommen wir nun zur ersten komplexen Grafikfunktion, dem Zeichnen einer Linie. Hierfür existieren viele verschiedene Algorithmen, von denen einige langsamer und speicherplatzsparender sind, andere schneller und speicherfressender. In meinem Buch *Amiga: Programmieren in Maschinensprache* können Sie ein Verfahren nachvollziehen, das sehr schnell arbeitet, weil für horizontale und vertikale Linien eigene Routinen programmiert wurden; dafür benötigt es jedoch wirklich einen großen Codeumfang. Beim

Portfolio ist die Geschwindigkeit nicht ganz so relevant, da zum einen Linien durch den kleinen Bildschirm bedingt nicht aus so vielen Grafikpunkten bestehen wie beim großen PC und zum anderen der Speicherbedarf an allererster Stelle steht.

Ich habe deshalb den sogenannten *Bresenham*-Algorithmus gewählt (nach seinem Erfinder benannt), der den Vorteil des geringen Platzbedarfs mit einer noch erträglichen Geschwindigkeit verknüpft, da ausschließlich Integer-Arithmetik zur Anwendung kommt. Eine Linie läßt sich nach der Gleichung

y = m * x + b

beschreiben, wobei m die Steigung und b den Offset (dies ist der y-Wert bei x=0) darstellt. Das Problem besteht nun darin, daß m in den allermeisten Fällen keine ganze Zahl darstellt, wodurch man theoretisch mit Fließkommarithmetik arbeiten müßte. Stellen Sie sich ein Blatt kariertes Papier vor, wobei jeder Eckpunkt eines Kästchens eine Koordinate auf dem Grafikschirm darstellt. Wenn Sie eine beliebige Linie auf dieses Papier malen, stellen Sie fest, daß diese nur in sehr wenigen Fällen direkt durch die Ecken der Kästchen geht, meistens schneidet sie die Kästchen irdendwo zwischen den Eckpunkten (Bild 2.3).

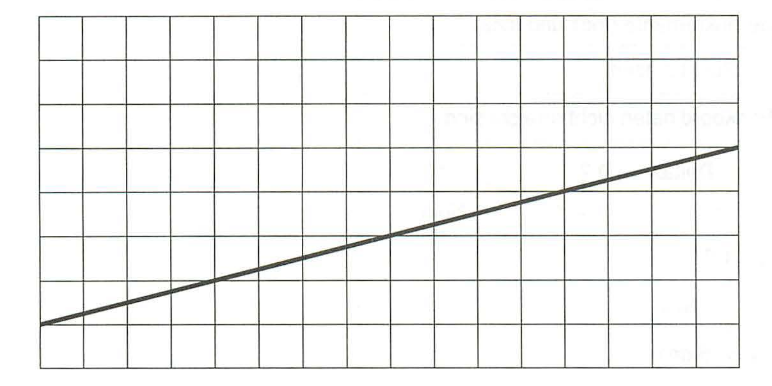


Bild 2.3: Ideale Linie

Nun, diese Zwischenwerte sind auf dem Papier kein Problem, der Computerbildschirm kennt aber keine »krummen« Koordinaten wie X = 3,5. Für den Computer müßte man daher jeden »krummen« Wert zunächst nach oben oder unten runden, bevor man ihn ausgeben kann, wodurch das berühmte Treppenmuster entsteht (Bild 2.4).

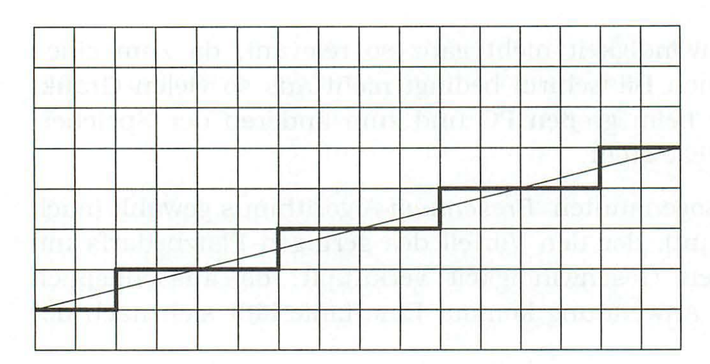


Bild 2.4: Linie auf Computerbildschirm

Berechnen der Absolutbeträge von DeltaX und DeltaY

Feststellen der Vorzeichen von DeltaX und DeltaY
=> Erhöhungswerte bestimmen (SignX und SignY)

DeltaX < DeltaY ?

Ja

Marke t = 1

DeltaX und DeltaY vertauschen

Nein

Marke t = 0

Berechnen des Startwertes

Berechnen der Inkremente (Inc1 und Inc2)

Punkt setzen

so lange Endkoordinaten nicht erreicht sind

DeltaX >= 0 ?

Ja

Marke t = 1 ?

Ja: x=x+SignX

Nein: y=y+SignY

d = d - Inc2

Nein

Marke t = 1 ?

Ja: y = y + SignY

Nein: x = x + SignX

d = d + Inc1

Punkt setzen

Bild 2.5: Bresenham-Algorithmus

Herr Bresenham erkannte, daß es paradox ist, zunächst Fließkommazahlen zu erzeugen, die anschließend wieder gerundet werden müssen. Sein Verfahren beruht auf der Idee, daß man das Auf- und Abrunden jedes Punktes allein durch die Steigung m bestimmen kann. Die Frage lautet: Nach wie vielen Schritten in x-Richtung muß ich auch einen Schritt in y-Richtung machen? Dazu wird aus der Steigung ein Wert bestimmt, von dem nach jedem Schritt in x-Richtung ein bestimmter zweiter Wert abgezogen wird. Ist das Ergebnis kleiner Null, muß auch ein Schritt in y-Richtung getätigt werden, worauf der erste Wert wieder hinzugezählt wird. Ist das Ergebnis jedoch nicht kleiner als Null, bleibt der y-Wert bestehen. Ist die Linie steiler als die Winkelhalbierende, kehrt sich die Fragestellung um, d.h. man fragt dann, nach wie vielen Schritten in y-Richtung muß auch ein Schritt in x-Richtung getätigt werden. In Bild 2.5 ist der Algorithmus grafisch dargestellt.

```
; LINE.ASM: Zieht eine Linie im Grafik-Bildschirm
;
; Autor: Frank Riemenschneider
;
; Eingabe:      AX = x-Koordinate (0-239) des Startpunktes
;               BX = y-Koordinate (0-63) des Startpunktes
;               CX = x-Koordinate (0-239) des Endpunktes
;               DX = y-Koordinate (0-63) des Endpunktes
;
; Ausgabe:      keine

DATASEG

        DeltaX  dw ?            ; Steigung in x-Richtung
        DeltaY  dw ?            ; Steigung in y-Richtung
        SignX   dw ?            ; Vorzeichen x-Richtung
        SignY   dw ?            ; Vorzeichen y-Richtung
        Inc1    dw ?            ; Inkrement 1
        Inc2    dw ?            ; Inkrement 2

CODESEG

PROC AsmLine
        push ax
        push bx
        push cx
        push dx
        push si
        push di
        push bp

        mov [DeltaX], cx        ; Steigung in x-Richtung berechnen
```

```
        mov [SignX], 1          ; positives Vorzeichen annehmen
        sub [DeltaX], ax
        jz LineLab8
        jg LineLab9             ; Steigung negativ?
        neg [DeltaX]            ; Ja -> DeltaX komplementieren
        mov [SignX], -1         ; negatives Vorzeichen in x-Richtung
        jmp short LineLab9
LineLab8:
        mov [SignX], 0          ; waagerechte Linie => Vorzeichen = 0
LineLab9:
        mov [DeltaY], dx        ; Steigung in y-Richtung berechnen
        mov [SignY], 1          ; positives Vorzeichen annehmen
        sub [DeltaY], bx
        jz LineLab10
        jg LineLab11            ; Steigung negativ?
        neg [DeltaY]            ; Ja -> DeltaY komplementieren
        mov [SignY], -1         ; negatives Vorzeichen in y-Richtung
        jmp short LineLab11
LineLab10:
        mov [SignY], 0
LineLab11:
        mov bp, 0               ; Richtungsflag löschen
        push ax                 ; Koordinate retten, um das Register zum
                                ; Arbeiten freizubekommen
        mov ax, [DeltaX]
        cmp ax, [DeltaY]        ; Größe der Steigungen in x- und y-Richtung
                                ; vergleichen
        jge LineLab12           ; Ja -> Steigungen nicht tauschen
        mov ax, [DeltaX]        ; Steigungen vertauschen
        xchg [DeltaY], ax
        mov [DeltaX], ax
        mov bp, 1               ; Richtungsflag setzen
LineLab12:
        mov ax, [DeltaY]        ; Steigung y-Richtung laden,
        shl ax, 1               ; mit 2 multiplizieren,
        sub ax, [DeltaX]        ; davon Steigung x-Richtung abziehen
        mov si, ax              ; Ergebnis ist der Startwert
        mov ax, [DeltaX]        ; Inkrement 2 berechnen
        shl ax, 1               ; als Doppeltem der Steigung x-Richtung
        mov [Inc2], ax
        mov ax, [DeltaY]        ; Inkrement 1 berechnen
        shl ax, 1               ; als Doppeltem der Steigung y-Richtung
        mov [Inc1], ax
        pop ax                  ; x-Koordinate Startpunkt wiederherstellen
        call AsmPlot            ; Punkt setzen
LineWhileLoop:
        cmp ax, cx              ; x-Koordinate Endpunkt erreicht?
        jne LineLab13           ; Nein -> weiter plotten
        cmp bx, dx              ; y-Koordinate Endpunkt erreicht?
        jne LineLab13           ; Nein -> weiter plotten
        jmp LineExit            ; Ja -> Beenden
LineLab13:
        cmp si, 0               ; Startwert >= 0?
        jl LineLab16            ; Nein -> weiter hinten
```

```
        cmp bp, 1               ; Flag = 1?
        jne LineLab14
        add ax, [SignX]         ; x-Koordinate erhöhen
        jmp short LineLab15
LineLab14:
        add bx, [SignY]         ; y-Koordinate erhöhen
LineLab15:
        sub si, [Inc2]          ; Startwert um Inkrement 2 erniedrigen
LineLab16:
        cmp bp, 1               ; Flag = 1?
        jne LineLab17           ; Nein -> weiter hinten
        add bx, [SignY]         ; y-Koordinate erhöhen
        jmp short LineLab18
LineLab17:
        add ax, [SignX]         ; x-Koordinate erhöhen
LineLab18:
        add si, [Inc1]          ; Startwert um Inkrement 1 erhöhen
        call AsmPlot            ; Punkt setzen
        jmp LineWhileLoop       ; Zurück zum Schleifenanfang

LineExit:
        pop bp
        pop di                  ; Alle Register wieder von den Bäumen,
        pop si                  ; äh, vom Stack herunterholen
        pop dx
        pop cx
        pop bx
        pop ax
        ret                     ; Back home

ENDP

; LINE.ASM:     Regelt die Parameterübernahme von Turbo Pascal aus für
;               die ASMLINE-Routine in Turbo Assembler
;
; Eingabe:      Parameter der LINE-Funktion auf dem Stack
;               Line( XAnf, YAnf, XEnde, YEnde:WORD );
; Ausgabe:      keine

CODESEG

PROC Line FAR XAnf:WORD, YAnf:WORD, XEnde:WORD, YEnde:WORD
        PUBLIC Line

        mov ax, [XAnf]          ; Anfangs- und Endkoordinaten in die
        mov bx, [YAnf]          ; entsprechenden Register bringen
        mov cx, [XEnde]
        mov dx, [YEnde]
; Koordinaten testen
        cmp ax, 0               ; XAnf < 0?
        jge LineLab1
        xor ax, ax
LineLab1:
        cmp ax, 239             ; XAnf > 239?
```

```
        jle LineLab2
        mov ax, 239
LineLab2:
        cmp bx, 0               ; YAnf < 0?
        jge LineLab3
        xor bx, bx
LineLab3:
        cmp bx, 63              ; YAnf > 63?
        jle LineLab4
        mov bx, 63
LineLab4:
        cmp cx, 0               ; XEnde < 0?
        jge LineLab5
        xor cx, cx
LineLab5:
        cmp cx, 239             ; XEnde > 239?
        jle LineLab6
        mov cx, 239
LineLab6:
        cmp dx, 0               ; YEnde < 0?
        jge LineLab7
        xor dx, dx
LineLab7:
        cmp dx, 63              ; YEnde > 63?
        jle LineLab7a
        mov dx, 63
LineLab7a:
        call AsmLine            ; Line-Routine aufrufen
        ret                     ; Back home

ENDP
```

2.5.5 Rechteck zeichnen

Mit Hilfe der *Line*-Routine ist das Zeichnen eines Rechtecks ein Kinderspiel, da diese nur viermal hintereinander mit den Eckpunkten als Start- und Zielkoordinaten aufgerufen werden muß.

```
; ASMBOX.ASM: Zeichnet ein Rechteck im Grafik-Bildschirm
;
; Autor: Frank Riemenschneider
;
; Eingabe:      AX = x-Koordinate (0-239) der oberen linken Ecke
;               BX = y-Koordinate (0-63) der oberen linken Ecke
;               CX = x-Koordinate (0-239) der unteren rechten Ecke
;               DX = y-Koordinate (0-63) der unteren rechten Ecke
;
; Ausgabe:      keine
```

```
DATASEG

X1      dw ?                ; x-Koordinate linke obere Ecke
X2      dw ?                ; x-Koordinate rechte untere Ecke
Y1      dw ?                ; y-Koordinate linke obere Ecke
Y2      dw ?                ; y-Koordinate rechte untere Ecke

CODESEG

PROC AsmBox

        push ax             ; Alle Register retten, die nicht schnell
        push bx             ; genug in den nächsten Stack kommen
        push cx
        push dx
        push di
        mov [X1], ax        ; Koordinaten abspeichern
        mov [Y1], bx
        mov [X2], cx
        mov [Y2], dx
        mov dx, [Y1]
        call AsmLine        ; obere Linie zeichnen
        mov ax, [X2]
        mov dx, [Y2]
        call AsmLine        ; rechte Linie zeichnen
        mov ax, [X1]
        mov bx, [Y2]
        call AsmLine        ; untere Linie zeichnen
        mov cx, [X1]
        mov bx, [Y1]
        call AsmLine        ; linke Linie zeichnen

BoxExit:
        pop di
        pop dx
        pop cx
        pop bx
        pop ax
        ret                 ; Back home

ENDP

; BOX.ASM:      Regelt die Parameterübernahme von Turbo Pascal aus für
;               die ASMBOX-Routine in Turbo Assembler
;
; Eingabe:      Parameter der Box-Funktion auf dem Stack
;               Box( Xol, Yol, Xru, Yru:WORD );
; Ausgabe:      keine

CODESEG

PROC Box FAR Xlo:WORD, Ylo:WORD, Xru:WORD, Yru:WORD
        PUBLIC Box
```

```
        mov ax, [Xlo]           ; Anfangs- und Endkoordinaten in die
        mov bx, [Ylo]           ; entsprechenden Register bringen
        mov cx, [Xru]
        mov dx, [Yru]
; Koordinaten testen
        cmp ax, 0               ; XAnf < 0?
        jge BoxLab1
        xor ax, ax
BoxLab1:
        cmp ax, 239             ; XAnf > 239?
        jle BoxLab2
        mov ax, 239
BoxLab2:
        cmp bx, 0               ; YAnf < 0?
        jge BoxLab3
        xor bx, bx
BoxLab3:
        cmp bx, 63              ; YAnf > 63?
        jle BoxLab4
        mov bx, 63
BoxLab4:
        cmp cx, 0               ; XEnde < 0?
        jge BoxLab5
        xor cx, cx
BoxLab5:
        cmp cx, 239             ; XEnde > 239?
        jle BoxLab6
        mov cx, 239
BoxLab6:
        cmp dx, 0               ; YEnde < 0?
        jge BoxLab7
        xor dx, dx
BoxLab7:
        cmp dx, 63              ; YEnde > 63?
        jle BoxLab8
        mov dx, 63
BoxLab8:
        cmp ax, cx              ; XAnf > XEnde?
        jle BoxLab9
        xchg ax, cx
BoxLab9:
        cmp bx, dx              ; YAnf > YEnde?
        jle BoxLab10
        xchg bx, dx
BoxLab10:
        call AsmBox             ; Box-Routine aufrufen
        ret                     ; Back home

ENDP
```

2.5.6 Ausgefülltes Rechteck zeichnen

Ebenso einfach ist das Zeichnen eines ausgefüllten Rechtecks, da hierfür ebenfalls nur die *Line*-Routine mehrmals hintereinander aufgerufen werden muß.

```
; BAR.ASM: Zeichnet ein gefülltes Rechteck im Grafik-Bildschirm
;
; Autor: Frank Riemenschneider
;
; Eingabe:      AX = x-Koordinate (0-239) der oberen linken Ecke
;               BX = y-Koordinate (0-63) der oberen linken Ecke
;               CX = x-Koordinate (0-239) der unteren rechten Ecke
;               DX = y-Koordinate (0-63) der unteren rechten Ecke
;
; Ausgabe:      keine

DATASEG

XB1     dw ?                    ; x-Koordinate linke obere Ecke
XB2     dw ?                    ; x-Koordinate rechte untere Ecke
YB1     dw ?                    ; y-Koordinate linke obere Ecke
YB2     dw ?                    ; y-Koordinate rechte untere Ecke

CODESEG

PROC AsmBlock

        push ax                 ; Alle Register retten, die nicht schnell
        push bx                 ; genug in den nächsten Stack kommen
        push cx
        push dx
        push di
        mov [XB1], ax           ; Koordinaten abspeichern
        mov [YB1], bx
        mov [XB2], cx
        mov [YB2], dx
        mov dx, [YB1]
BlockLoop:
        call AsmLine            ; oberste Linie zeichnen
        cmp bx, [YB2]           ; Ende erreicht?
        je BlockExit            ; Ja -> Beenden
        inc bx                  ; Start- und Endkoordinaten der Linie erhöhen
        inc dx
        jmp BlockLoop           ; Nächste Linie zeichnen

BlockExit:
        pop di
        pop dx
        pop cx
        pop bx
        pop ax
        ret                     ; Back home
```

```
ENDP

; BAR.ASM:       Regelt die Parameterübernahme von Turbo Pascal aus
;                für die ASMBLOCK-Routine in Turbo Assembler
;
; Eingabe:       Parameter der Block-Funktion auf dem Stack
;                Bar( Xlo, Ylo, Xru, Yru:WORD );
; Ausgabe:       keine

CODESEG

PROC Bar Xlo:WORD, Ylo:WORD, Xru:WORD, Yru:WORD
        PUBLIC Bar

        mov ax, [Xlo]             ; Anfangs- und Endkoordinaten in die
        mov bx, [Ylo]             ; entsprechenden Register bringen
        mov cx, [Xru]
        mov dx, [Yru]
; Koordinaten testen
        cmp ax, 0                 ; XAnf < 0?
        jge BlockLab1
        xor ax, ax
BlockLab1:
        cmp ax, 239               ; XAnf > 239?
        jle BlockLab2
        mov ax, 239
BlockLab2:
        cmp bx, 0                 ; YAnf < 0?
        jge BlockLab3
        xor bx, bx
BlockLab3:
        cmp bx, 63                ; YAnf > 63?
        jle BlockLab4
        mov bx, 63
BlockLab4:
        cmp cx, 0                 ; XEnde < 0?
        jge BlockLab5
        xor cx, cx
BlockLab5:
        cmp cx, 239               ; XEnde > 239?
        jle BlockLab6
        mov cx, 239
BlockLab6:
        cmp dx, 0                 ; YEnde < 0?
        jge BlockLab7
        xor dx, dx
BlockLab7:
        cmp dx, 63                ; YEnde > 63?
        jle BlockLab8
        mov dx, 63
BlockLab8:
        cmp ax, cx                ; XAnf > XEnde?
        jle BlockLab9
```

```
        xchg ax, cx
BlockLab9:
        cmp bx, dx                  ; YAnf > YEnde?
        jle BlockLab10
        xchg bx, dx
BlockLab10:
        call AsmBlock               ; Block-Routine aufrufen
        ret                         ; Back home

ENDP
```

2.5.7 Ellipse(nabschnitt) zeichnen

Wir wollen nun die Gebilde verlassen, die ausschließlich aus Geraden bestehen, und uns den Kreisen und Ellipsen zuwenden, deren Aufbau den einer Linie an Komplexität noch weit übertrifft.

Mathematisch gesehen gibt es drei Möglichkeiten, eine Ellipse zu beschreiben: Die kartesischen Koordinaten, die Zylinderkoordinaten und die Kugelkoordinaten. Meistens wird mit den Zylinderkoordinaten gerechnet. Wenn man den Mittelpunkt der Ellipse durch die Koordinaten XM und YM beschreibt, den Radius in x-Richtung mit XM und den in y-Richtung mit YM, und schließlich den Winkel W einführt, kann man die Ellipse als die Punktmenge definieren, welche die folgenden beiden Gleichungen erfüllt:

$X = XM + XRxCOS(W) \qquad 0 \text{ Grad} < W < 360 \text{ Grad}$
$Y = YM + YRxSIN(W) \qquad 0 \text{ Grad} < W < 360 \text{ Grad}$

Falls XR und YR gleich sind, liegt der Sonderfall eines Kreises vor. Man braucht also tatsächlich nur eine Routine, um sowohl Kreise als auch Ellipsen zeichnen zu können.

Das übliche Verfahren, um in den Zylinderkoordinaten eine Ellipse zu zeichnen, besteht einfach darin, für jeden Winkel W die beiden obigen Gleichungen auszuwerten, also die Sinus- und Cosinus-Werte zu berechnen, mit den Radien zu multiplizieren und das Ergebnis in eine Integer-Zahl umzuwandeln, die dann zu den Mittelpunktkoordinaten addiert wird. Ein Problem besteht darin, daß bei einer idealen Ellipse zwei nebeneinanderliegende Winkel W unendlich dicht zusammenliegen, was theoretisch unendlich viele Berechnungsschritte erforderlich macht. Viele Programme helfen sich damit, daß nur Punkte gewisser Winkel berechnet werden und diese dann durch Linien verbunden werden. Je größer die Schrittweite ist,

desto größer ist natürlich die Zeichengeschwindigkeit, desto größer aber auch die Abweichung von einer idealen Ellipse.

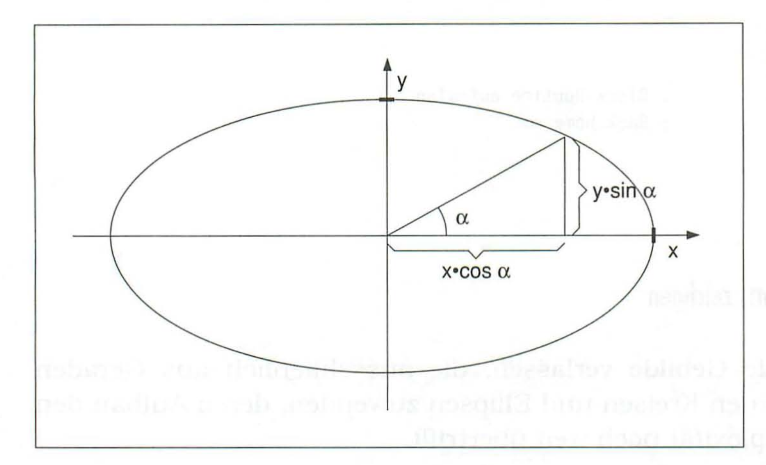


Bild 2.6: Aufbau einer Ellipse

Es ist jedoch klar, daß der Portfolio keine ideale Ellipse darstellen kann, da seine Grafikauflösung nicht unendlich groß ist, sondern nur 240x64 Punkte beträgt. Die genaueste mögliche Ellipse erhält man, wenn zwei nebeneinanderliegende x- und y-Werte jeweils nicht mehr als einen Grafikpunkt auseinanderliegen. Dieses Kriterium kann man für einen bestimmten Computer einführen, um den Begriff »Genauigkeit« zu definieren.

In der Praxis heißt dies, daß man eine Schrittweite von ca. 0,1 Grad nehmen muß, um dieser Forderung zu entsprechen. Leider wird die Zeichengeschwindigkeit dadurch aber unakzeptabel hoch. Daher überlegte man sich ein »Optimierungsverfahren« (dieses verdient den Namen nicht, deshalb habe ich es in Anführungszeichen gesetzt), wobei mathematische Beziehungen zwischen der Sinus- und Cosinus-Funktion ausgenutzt werden:

1: SIN(W) = SIN(180 Grad – W) = –SIN(180 Grad + W) = –SIN(360 Grad – W)
2: COS(W) = COS(360 Grad – W) = –COS(180 Grad – W) = –COS(180 Grad + W)

Durch diese Tatsache ist es möglich, mit einem berechneten Wertepaar (X,Y) jeweils in jedem Quadranten einen Punkt zu setzen, da sich nur die Vorzeichen unterscheiden. Wenn wir den Term XRxCOS(W) mit XOFF und YRxSIN(W) mit YOFF bezeichnen, ergeben sich für die Koordinaten der einzelnen Quadranten folgende Zusammenhänge:

1. Quadrant (0- 90 Grad):   X = XM + XOFF; Y = YM – YOFF
2. Quadrant (90-180 Grad):   X = XM + XOFF; Y = YM + YOFF
3. Quadrant (180-270 Grad):   X = XM – XOFF; Y = YM + YOFF
4. Quadrant (270-360 Grad):   X = XM – XOFF; Y = YM – YOFF

Durch diese Methode kann man die Rechenzeit ca. um den Faktor 4 verringern. Leider kann eine Zeichenroutine dieses Musters aber keine beliebigen Ausschnitte einer Ellipse zeichnen.

Trotz dieses Kunstgriffs ist es aber nicht gelungen, die Rechenzeit soweit zu verkürzen, daß man den Ellipsenaufbau optisch nicht mehr mitverfolgen kann, was für mich das Kriterium von Geschwindigkeit ist. Das eigentliche Problem besteht weiterhin in der Verwendung von Fließkommazahlen, die ja eigentlich unsinnig sind, wie wir schon im Abschnitt über die Linien gesehen haben. Die Frage ist, wie man Sinus- und Cosinus-Werte berechnen kann, ohne Fließkommazahlen zu benutzen. Die Antwort: überhaupt nicht. Ich möchte Ihnen daher ein Verfahren vorstellen, das ich 1987 für den C64 entwickelt habe (dieser Rechner besaß einen 6502-Prozessor, der mit 1 (!) MHz getaktet wurde) und 1989 auf den Amiga übertragen habe (68000-Prozessor mit 8 MHz), wo die Ellipsen-Routine am Ende 5000 Grafikpunkte pro Sekunde zeichnete. Nun, solche Ergebnisse sind auf dem Portfolio allein schon wegen des langsamen LCD-Kontrollers nicht zu erwarten, die C64-Ergebnisse werden jedoch allemal erreicht.

Die Idee des Verfahrens besteht darin, die Sinus- und Cosinus-Werte in einer Integer-Tabelle anzulegen. Dies ist unmittelbar nicht möglich, da sich die Werte ja nur im Bereich zwischen 0 und 1 bewegen. Wenn man jeden dieser Werte aber mit einer hinreichend großen Zahl multipliziert, steht der Umwandlung ins Integer-Format nichts mehr im Wege. Um mit dem Variablentyp *Byte* arbeiten zu können, wodurch die maximale Geschwindigkeit erzielt wird, werden wir den Wert 255 nehmen. Daher gilt für einen Tabelleneintrag:

```
Wert = INT(COS(W)x255)
```

Um den Offset zu erhalten, müssen wir

```
XOFF = XM + XRx(Wert/255)
```

rechnen. An dieser Stelle müssen wir uns wieder mit dem Problem Genauigkeit auseinandersetzen. Wie viele Werte soll die Tabelle enthalten und in welcher Schrittweite müssen diese angelegt werden?

Wir benötigen die Werte von 0 Grad < W < 90 Grad, die auch für die anderen drei Quadranten verwendet werden können (s.o.). Unsere Forderung ist, daß zwei benachbarte Werte maximal um die Zahl 1 differieren dürfen, um die größtmögliche Genauigkeit zu erhalten. Hier hilft nur ausprobieren: Es zeigt sich, daß die Schrittweite 1/394 die größte mögliche ist, die diese Forderung erfüllt. Eine geringere Schrittweite bringt einen Zuwachs an Rechenzeit, eine größere einen Verlust an Genauigkeit. Wir haben nach dem Grundsatz »so grob wie möglich, so fein wie nötig« die optimale Schrittweite erhalten. Unsere Tabellenwerte können wir nun in einer Schleife berechnen:

```
FOR I = 0 TO 394 : WERT(I) = 255xINT(COS(90/394)xI)) : NEXT I
```

Damit haben wir es geschafft, den Übergang vom Winkel W in eine Schleife von Integer-Werten zu erreichen. Die kritischste Stelle der Genauigkeit liegt übrigens bei 90 Grad, da der Cosinus dort seine größte Steigung aufweist. Der x-Offset läßt sich nun auf einfachste Weise berechnen: Wir müssen in einer Schleife von 0 bis 394 die jeweiligen Tabellenwerte auslesen, mit dem x-Radius multiplizieren und durch 255 teilen:

```
FOR I = 0 TO 394 : X = XM + (XRxWert(I))/255 : NEXT I
```

Hier der Beweis: Wenn wir die Berechnungsgrundlage für die Tabellenwerte einsetzen, ergibt sich folgendes:

```
FOR I = 0 TO 394 : X = XM + (XRx255xINT(COS(90/394)xI))/255 : NEXT I
```

Dies kann man aber auch so ausdrücken:

```
FOR I = 0 TO 90 : X = XM + XRxINT(COS(I)) : NEXT I
```

Das einzige Problem besteht darin, daß nur Radien bis einschließlich des Werts 255 zugelassen sind, da bis zu diesem Zeitpunkt der Ausdruck

```
XOFF = (XRxWERT(I))/255
```

nicht größer als 1 werden kann, was für die erforderliche Genauigkeit ja notwendig ist. Mit dieser Einschränkung kann man aber leben, wenn man bedenkt, daß der maximale Durchmesser in x-Richtung nur 240 Punkte und in y-Richtung sogar nur 64 Punkte beträgt. Für die Sinus-Werte muß übrigens keine eigene Tabelle angelegt werden, da für diesen gilt:

```
SIN(W) = COS(90 Grad - W)
```

Man liest die Tabelle also einmal von vorne für die Cosinus-Werte und einmal von hinten für die Sinus-Werte. Hier nun der Quelltext der Routine, die noch zusätzlich zwischen einer ganzen Ellipse und einem Ausschnitt unterscheidet.

```
; ASMCURVE.ASM: Zeichnet einen Ellipsenbogen
;
; Autor: Frank Riemenschneider
;
; Eingabe:     AX = x-Koordinate Ellipsenmittelpunkt (0-239)
;              BX = y-Koordinate Ellipsenmittelpunkt (0-63)
;              CX = Radius in x-Richtung (0-255)
;              DX = Radius in y-Richtung (0-255)
;              ES = Anfangswinkel in Grad (0-359°)
;              SI = Endwinkel in Grad (0-360°)
; Ausgabe: keine

DATASEG

; Benötigte Variable
x           dw ?   ; x-Koordinate
y           dw ?   ; y-Koordinate
xr          dw ?   ; x-Radius
yr          dw ?   ; y-Radius
xoff        dw ?   ; x-Offset
yoff        dw ?   ; y-Offset
zaehl1      dw ?   ; Zähler 1
zaehl2      dw ?   ; Zähler 2
xm          dw ?   ; x-Koordinate Mittelpunkt
anqua       dw ?   ; Anfangsquadrant
endqua      dw ?   ; Endquadrant
anpunkt     dw ?   ; Anfangspunkt
endpunkt    dw ?   ; Endpunkt
offtab1     db 400 dup (?)     ; Offset-Tabellen
offtab2     db 400 dup (?)

extrn costab : DWORD
extrn ArcStartX : DWORD
extrn ArcStartY : DWORD
extrn ArcEndX : DWORD
extrn ArcEndY : DWORD

CODESEG

; Koordinate des Start- und Endpunktes des Ellipsenbogens berechnen
MACRO CalcPoints StartQuad, EndQuad
IFIDNI <StartQuad>, <1>
   mov bx, 394
   sub bx, [anpunkt]
ENDIF
IFIDNI <StartQuad>, <2>
   mov bx, [anpunkt]
```

```
ENDIF
IFIDNI <StartQuad>, <3>
   mov bx, 394
   sub bx, [anpunkt]
ENDIF
IFIDNI <StartQuad>, <4>
   mov bx, [anpunkt]
ENDIF
   xor ch, ch
   mov dh, ch
   mov cl, [offtab1+bx]
   mov dl, [offtab2+bx]
IFIDNI <StartQuad>, <1>
   add cx, [xm]
   mov [WORD ArcStartX], cx
   mov cx, [y]
   sub cx, dx
   mov [WORD ArcStartY], cx
ENDIF
IFIDNI <StartQuad>, <2>
   add cx, [xm]
   mov [WORD ArcStartX], cx
   add dx, [y]
   mov [WORD ArcStartY], dx
ENDIF
IFIDNI <StartQuad>, <3>
   mov ax, [xm]
   sub ax, cx
   mov [WORD ArcStartX], ax
   add dx, [y]
   mov [WORD ArcStartY], dx
ENDIF
IFIDNI <StartQuad>, <4>
   mov ax, [xm]
   sub ax, cx
   mov [WORD ArcStartX], ax
   mov cx, [y]
   sub cx, dx
   mov [WORD ArcStartY], cx
ENDIF
IFIDNI <EndQuad>, <1>
   mov bx, 394
   sub bx, [endpunkt]
ENDIF
IFIDNI <EndQuad>, <2>
   mov bx, [endpunkt]
ENDIF
IFIDNI <EndQuad>, <3>
   mov bx, 394
   sub bx, [endpunkt]
ENDIF
IFIDNI <EndQuad>, <4>
   mov bx, [endpunkt]
ENDIF
```

```
   xor ch, ch
   mov dh, ch
   mov cl, [offtab1+bx]
   mov dl, [offtab2+bx]
IFIDNI <EndQuad>, <1>
   add cx, [xm]
   mov [WORD ArcEndX], cx
   mov cx, [y]
   sub cx, dx
   mov [WORD ArcEndY], cx
ENDIF
IFIDNI <EndQuad>, <2>
   add cx, [xm]
   mov [WORD ArcEndX], cx
   add dx, [y]
   mov [WORD ArcEndY], dx
ENDIF
IFIDNI <EndQuad>, <3>
   mov ax, [xm]
   sub ax, cx
   mov [WORD ArcEndX], ax
   add dx, [y]
   mov [WORD ArcEndY], dx
ENDIF
IFIDNI <EndQuad>, <4>
   mov ax, [xm]
   sub ax, cx
   mov [WORD ArcEndX], ax
   mov cx, [y]
   sub cx, dx
   mov [WORD ArcEndY], cx
ENDIF
ENDM

; Quadranten zeichnen
MACRO QuadLoop Quad, Index1, Index2
   xor ch, ch
   mov dh, ch
LoopQ&Quad:
   mov bx, [zaehl&Index1]
   mov cl, [offtab1+bx]         ; x-Offset holen
   mov dl, [offtab2+bx]         ; y-Offset holen
   mov ax, [xm]                 ; x-Koordinate Mittelpunkt holen
IFIDNI <Quad>, <1>
   add ax, cx                   ; x = xm + xoff
   mov bx, [y]                  ; y-Koordinate holen
   sub bx, dx                   ; y = y - yoff
ENDIF
IFIDNI <Quad>, <2>
   add ax, cx                   ; x = xm + xoff
   mov bx, [y]
   add bx, dx                   ; y = y + yoff
ENDIF
IFIDNI <Quad>, <3>
```

```
   sub ax, cx                ; x = xm - xoff
   mov bx, [y]
   add bx, dx                ; y = y + yoff
ENDIF
IFIDNI <Quad>, <4>
   sub ax, cx                ; x = xm - xoff
   mov bx, [y]
   sub bx, dx                ; y = y - yoff
ENDIF
   call AsmPlot              ; Punkt setzen
   mov ax, [zaehl&Index1]
   cmp ax, [zaehl&Index2]    ; alle Punkte gesetzt?
   je ExitLoopQ&Quad         ; Ja -> Beenden
   inc ax                    ; Zähler erhöhen
   mov [zaehl&Index1], ax
   jmp LoopQ&Quad
ExitLoopQ&Quad:
   ret
ENDM

; Koordinaten holen
PROC koor NEAR
        mov [xm], ax         ; x-Koordinate Mittelpunkt
        mov [y], bx          ; y-Koordinate Mittelpunkt
        mov [xr], cx         ; Radius x-Richtung
        mov [yr], dx         ; Radius y-Richtung
        mov ax, es           ; Anfanswinkel in AX
        call rechne          ; Anfangswinkel in Anfangsquadranten und
                             ; Anfangspunkt umrechnen
        mov [anqua], ax      ; Anfangsquadrant
        mov [anpunkt], bx    ; Anfangspunkt
        mov ax, si           ; Endwinkel in AX
        call rechne          ; Endwinkel in Endquadranten und
                             ; Endpunkt umrechnen
        mov [endqua], ax     ; Endquadrant
        mov [endpunkt], bx   ; Endpunkt
        ret
; Umrechnung: Winkel -> Quadrant und Punkt
rechne: push ax              ; wird später noch gebraucht
        mov cx, 90
        div cl               ; Winkel / 90
        xor ah, ah           ; Divisionsrest löschen
                             ; => INT(Winkel / 90)
        inc al               ; Quadrant = INT(Winkel / 90) + 1
        mov dx, ax           ; Quadrant retten
        dec al
        mul cl               ; (Quadrant - 1) x 90
        pop bx               ; Winkel holen
        push dx
        sub bx, ax           ; Teilwinkel = Winkel - (Quadrant - 1) x 90,
                             ; 0° <= Teilwinkel < 90°
        mov ax, bx
        mov cx, 395
        mul cx               ; Teilwinkel x 395, Ergebnis in DX:AX
```

```
        mov cx, 90
        div cx                  ; Punkt = (Teilwinkel x 395) / 90°
        mov bx, ax              ; Werte in die richtigen Register bringen
        pop ax
        ret                     ; Back to happiness

ENDP

; Berechnung von AL x BX / 256
PROC Mult NEAR
        xchg ax, bx
        mul bl                  ; AL x BX
REPT 8
        shr ax, 1               ; AX 8 mal rechts schieben => AX / 256
ENDM
        ret
ENDP

; Ersten Quadranten zeichnen
PROC erster NEAR
        mov [zaehl1], ax
        mov [zaehl2], bx
        mov bx, 394             ; Zähler setzen
        sub bx, [zaehl1]        ; und invertieren
        mov [zaehl1], bx
        mov bx, 394
        sub bx, [zaehl2]
        mov [zaehl2], bx
        QuadLoop 1, 2, 1
ENDP

; Zweiten Quadranten zeichnen
PROC zweiter NEAR
        mov [zaehl1], ax        ; Zähler setzen
        mov [zaehl2], bx
        QuadLoop 2, 1, 2
ENDP

; Dritten Quadranten zeichnen
PROC dritter NEAR
        mov [zaehl1], ax        ; Zähler setzen
        mov [zaehl2], bx
        mov ax, 394
        sub ax, [zaehl1]        ; und invertieren
        mov [zaehl1], ax
        mov bx, 394
        sub bx, [zaehl2]
        mov [zaehl2], bx
        QuadLoop 3, 2, 1
ENDP
```

```
; Vierten Quadranten zeichnen
PROC vierter NEAR
        mov [zaehl1], ax        ; Zähler setzen
        mov [zaehl2], bx
        QuadLoop 4, 1, 2
ENDP

PROC AsmCurve

; Register retten
        push ax
        push bx
        push cx
        push dx
        push es
        push si
        push di

        call koor               ; Koordinaten zuweisen
        cmp [anqua], 1          ; Start im 1. Quadranten?
        jne ausschnitt          ; mit erstem Punkt
        cmp [anpunkt], 0        ; Nur Ellipsenabschnitt zeichnen?
        jne ausschnitt
        cmp [endqua], 5         ; Ende wieder im 1. Quadranten?
        jne ausschnitt          ; nur Ellipsenabschnitt
        cmp [endpunkt], 0       ; Komplette Ellipse zeichnen?
        jne ausschnitt          ; Nein -> Ellipsenabschnitt zeichnen
        je ganz                 ; Ja -> komplette Ellipse zeichnen
ausschnitt:
        jmp aussch              ; Sprung zum Zeichnen eines Ellipsenabschnitts

; komplette Ellipse zeichnen
ganz:
        mov bx, [xm]            ; x-Koordinate Mittelpunkt in x merken
        push ax
        mov ax, [y]             ; Anfangs- und Endpunktskoordinaten berechnen
        sub ax, [yr]            ; und speichern
        mov [WORD ArcStartX], bx
        mov [WORD ArcEndX], bx
        mov [WORD ArcStartY], ax
        mov [WORD ArcEndY], ax
        pop ax
        mov [x], bx
        mov [zaehl2], 0         ; Beide Zähler setzen
        mov [zaehl1], 394
CircleLoop:
        mov bx, [zaehl1]
        xor ah, ah
        mov al, [BYTE costab+bx]
                                ; Cosinus-Wert in AX
        mov bx, [xr]            ; x-Radius in BX
        call mult               ; AL*xr/256 berechnen
        mov [xoff], ax          ; = x-offset
        mov bx, [zaehl2]        ; Zähler2
```

```
        xor ah, ah
        mov al, [BYTE costab+bx]
                                ; Sinus-Wert holen
        mov bx, [yr]            ; y-Radius
        call mult               ; AL*yr/256 berechnen
        mov [yoff], ax          ; = y-offset
        mov ax, [xm]            ; x-Koordinate Mittelpunkt in AX
        add ax, [xoff]          ; x = xm + xoff
        mov bx, [y]             ; y-Koordinate in BX
        add bx, [yoff]          ; y = y + yoff
        mov cx, bx              ; y + yoff in CX retten
        call AsmPlot            ; 1. Punkt setzen
        mov bx, [y]             ; y-Koordinate in BX
        sub bx, [yoff]          ; y = y - yoff
        mov dx, bx              ; y - yoff in DX retten
        call AsmPlot            ; 2. Punkt setzen
        mov ax, [xm]            ; x-Koordinate Mittelpunkt laden
        sub ax, [xoff]          ; x = xm - xoff
        mov bx, cx              ; y + yoff laden
        call AsmPlot            ; 3. Punkt setzen
        mov bx, dx              ; y - yoff laden
        call AsmPlot            ; 4. Punkt setzen
        inc [zaehl2]            ; Zähler 1 erhöhen
        dec [zaehl1]            ; Zähler 2 erniedrigen
        jne CircleLoop          ; nächsten 4 Punkte berechnen
        jmp CircleExit1         ; Beide Zähler gleich -> Beenden

; Ellipsenabschnitt zeichnen
aussch:
        mov [zaehl1], 395       ; Beide Zähler laden
        mov [zaehl2], 0
ausloop:
        mov bx, [zaehl1]        ; Tabelle der Offsets in x- und y-Richtung
        xor ah, ah              ; aufbauen
        mov al, [BYTE costab+bx]
        mov bx, [xr]
        call mult
        mov bx, [zaehl2]
        mov [offtab1+bx], al
        xor ah, ah
        mov al, [BYTE costab+bx]
        mov bx, [yr]
        call mult
        mov bx, [zaehl2]
        mov [offtab2+bx], al
        inc [zaehl2]
        dec [zaehl1]
        jne ausloop
        mov ax, [xm]
        mov [x], ax
        mov bx, [anqua]         ; Anfangsquadranten laden
        shl bx, 1
        shl bx, 1               ; anqua x 4
        add bx, [endqua]        ; Endquadranten dazu addieren
```

```
        sub bx, 5               ; 0 <= Indexwert <= 15
        shl bx, 1               ; Indexwert x 8
        shl bx, 1
        shl bx, 1
        mov ax, OFFSET jumptab
        add ax, bx
        jmp ax

jumptab:
        call s1e1               ; Diese Sprungtabelle ist nötig, da
                                ; Assembler-Routinen in Turbo-Pascal
        jmp CircleExit1         ; keine typisierten Konstanten verwenden
        nop                     ; dürfen. Sonst könnte man dies wesentlich
        nop                     ; kürzer programmieren.
        call s1e2               ; Die NOPs dienen nur dazu, die Distanz
        jmp CircleExit1         ; zwischen den Sprungzielen auf 8 Byte
        nop                     ; aufzuweiten, damit die Sprungberechnung
        nop                     ; einfacher wird.
        call s1e3
        jmp CircleExit1
        nop
        nop
        call s1e4
        jmp CircleExit1
        nop
        nop
        call s2e1
        jmp CircleExit1
        nop
        nop
        call s2e2
        jmp CircleExit1
        nop
        nop
        call s2e3
        jmp CircleExit1
        nop
        nop
        call s2e4
        jmp CircleExit1
        nop
        nop
        call s3e1
        jmp CircleExit1
        nop
        nop
        call s3e2
        jmp CircleExit1
        nop
        nop
        call s3e3
        jmp CircleExit1
        nop
        nop
```

```
        call s3e4
        jmp CircleExit1
        nop
        nop
        call s4e1
        jmp CircleExit1
        nop
        nop
        call s4e2
        jmp CircleExit1
        nop
        nop
        call s4e3
        jmp CircleExit1
        nop
        nop
        call s4e4
        jmp CircleExit1


; Start im 1., Ende im 1. Quadranten
s1e1:
        CalcPoints 1, 1
        mov ax, [anpunkt]
        mov bx, [endpunkt]
        cmp ax, bx
        jle s1e11
        mov bx, 394
        call erster
        xor ax, ax
        jmp s2e1a
s1e11:  mov bx, [endpunkt]
        jmp erster

; Start im 1., Ende im 2. Quadranten
s1e2:
        CalcPoints 1, 2
        mov ax, [anpunkt]
s1e2b:  mov bx, 394
        call erster
        xor ax, ax
s1e2a:  mov bx, [endpunkt]
        jmp zweiter

; Start im 1., Ende im 3. Quadranten
s1e3:
        CalcPoints 1, 3
        mov ax, [anpunkt]
s1e3c:  mov bx, 394
        call erster
        xor ax, ax
s1e3a:  mov bx, 394
        call zweiter
        xor ax, ax
```

```
s1e3b:  mov bx, [endpunkt]
        jmp dritter

; Start im 1., Ende im 4. Quadranten
s1e4:
        CalcPoints 1, 4
        mov ax, [anpunkt]
s1e4c:  mov bx, 394
        call erster
        xor ax, ax
s1e4a:  mov bx, 394
        call zweiter
        xor ax, ax
s1e4b:  mov bx, 394
        call dritter
        xor ax, ax
s1e4d:  mov bx, [endpunkt]
        jmp vierter

; Start im 2., Ende im 1. Quadranten
s2e1:
        CalcPoints 2, 1
        mov ax, [anpunkt]
s2e1a:  mov bx, 394
        call zweiter
        xor ax, ax
s2e1b:  mov bx, 394
        call dritter
        xor ax, ax
s2e1c:  mov bx, 394
        call vierter
        xor ax, ax
        jmp s1e11

; Start im 2., Ende im 2. Quadranten
s2e2:
        CalcPoints 2, 2
        mov ax, [anpunkt]
        mov bx, [endpunkt]
        cmp ax, bx
        jle s2e22
        mov bx, 394
        call zweiter
        xor ax, ax
        jmp s3e2a
s2e22:  jmp s1e2a

; Start im 2., Ende im 3. Quadranten
s2e3:
        CalcPoints 2, 3
        mov ax, [anpunkt]
        jmp s1e3a

; Start im 2., Ende im 4. Quadranten
```

```
s2e4:
        CalcPoints 2, 4
        mov ax, [anpunkt]
        jmp s1e4a

; Start im 3., Ende im 1. Quadranten
s3e1:
        CalcPoints 3, 1
        mov ax, [anpunkt]
        jmp s2e1b

; Start im 3., Ende im 2. Quadranten
s3e2:
        CalcPoints 3, 2
        mov ax, [anpunkt]
s3e2a:  mov bx, 394
        call dritter
        xor ax, ax
s3e2b:  mov bx, 394
        call vierter
        xor ax, ax
        jmp s1e2b

; Start im 3., Ende im 3. Quadranten
s3e3:
        CalcPoints 3, 3
        mov ax, [anpunkt]
        mov bx, [endpunkt]
        cmp ax, bx
        jle s3e33
        mov bx, 394
        call dritter
        xor ax, ax
        jmp s4e3a
s3e33:  jmp s1e3b

; Start im 3., Ende im 4. Quadranten
s3e4:
        CalcPoints 3, 4
        mov ax, [anpunkt]
        jmp s1e4b

; Start im 4., Ende im 1. Quadranten
s4e1:
        CalcPoints 4, 1
        mov ax, [anpunkt]
        jmp s2e1c

; Start im 4., Ende im 2. Quadranten
s4e2:
        CalcPoints 4, 2
        mov ax, [anpunkt]
        jmp s3e2b
```

```
; Start im 4., Ende im 3. Quadranten
s4e3:
        CalcPoints 4, 3
        mov ax, [anpunkt]
s4e3a:  mov bx, 394
        call vierter
        xor ax, ax
        jmp s1e3c

; Start im 4., Ende im 4. Quadranten
s4e4:
        CalcPoints 4, 4
        mov ax, [anpunkt]
        mov bx, [endpunkt]
        cmp ax, bx
        jle s4e44
        mov bx, 394
        call vierter
        xor ax, ax
        jmp s1e4c
s4e44:  jmp s1e4d

CircleExit1:
        pop di                  ; Alle Registerinhalte wieder
        pop si                  ; laden
        pop es
        pop dx
        pop cx
        pop bx
        pop ax
        ret                     ; Back home

ENDP


; CURVE.ASM:    Regelt die Parameterübernahme von Turbo Pascal aus für
;               die ASMCURVE-Routine in Turbo Assembler
;
; Autor: Frank Riemenschneider
;
; Eingabe:      Parameter der CURVE-Funktion auf dem Stack
;               Circle( XM, YM, XR, YR, AnfWinkel, EndWinkel:WORD );
; Ausgabe:      keine

CODESEG

PROC Curve FAR XMP:WORD, YMP:WORD, XRA:WORD, YRA:WORD, AnfWinkel:WORD, EndWinkel:WORD
        PUBLIC Curve

        mov ax, [XMP]           ; Mittelpunkt-Koordinaten
        mov bx, [YMP]
        mov cx, [XRA]           ; Radius in x- und y-Richtung
        mov dx, [YRA]           ; Koordinaten prüfen
        cmp ax, 0               ; x-Koordinate < 0?
```

```
        jl CircleExit
        cmp ax, 239             ; x-Koordinate > 239?
        jg CircleExit
        cmp bx, 0               ; y-Koordinate < 0?
        jl CircleExit
        cmp bx, 63              ; y-Koordinate > 63?
        jg CircleExit
        mov si, [AnfWinkel]     ; Anfangswinkel laden
        cmp si, 0               ; Anfangswinkel < 0°?
        jl CircleExit
        cmp si, 359             ; Anfangswinkel > 359°?
        jg CircleExit
        mov es, si
        mov si, [EndWinkel]     ; Endwinkel laden
        cmp si, 0               ; Endwinkel < 0°?
        jl CircleExit
        cmp si, 360             ; Endwinkel > 360°?
        jg CircleExit
        call AsmCurve           ; Curve-Routine aufrufen
CircleExit:
        ret                     ; Back home

ENDP
```

2.5.8 Tortenstück zeichnen

Im Zeitalter gestreßter Manager werden Präsentationsgrafiken immer wichtiger, damit diese die aktuellen Umsätze mit einem Blick erfassen können, ohne vorher endlose Zahlenkolonnen durchforstet zu haben. Eine beliebte Darstellungsart ist dabei die Tortengrafik. Um ein zweidimensionales Tortenstück zu zeichnen, sind »nur« zwei Linien und ein Ellipsenabschnitt nötig – kein Problem also, wie es scheint. Doch wie so oft liegt der Teufel im Detail: Während die Ellipsenroutine mit Winkeln und Radien rechnet, benötigt die Linien-Routine kartesische Koordinaten für ihren Start- und Endpunkt. Nun, auch dies scheint zunächst kein Problem zu sein, da man die äußeren beiden Punkte der Ellipse über die Sinus- bzw. Cosinus-Funktion ermitteln kann (s.o.). Leider können diese beiden Funktionen aber so ungenau arbeiten, daß eine Verschiebung von einem Grafikpunkt im Bereich des Möglichen liegt. Die anschließend gezeichneten Linien enden also nicht unmittelbar an der Ellipse, sondern einen Punkt daneben. Dies wäre auch nocht nicht so schlimm, wenn man nicht das Tortenstück mit Farbe füllen möchte. Dann nämlich nimmt das Unglück seinen Lauf: Da die Fläche nicht geschlossen ist, wird der gesamte Bildschirm gefüllt, oder/und das Programm stürzt ab. Deshalb

haben die Pascal-Entwickler in weiser Voraussicht die *Slice*-Routine programmiert, die ein Tortenstück zeichnet. Hier nun die entsprechende Routine für die Portfolio-Unit.

```
; ASMPIESLICE.ASM: Zeichnet ein 2D-Kuchenstück, d.h. einen Ellipsenbogen
; und Linien vom Mittelpunkt zum Anfangs- und Endpunkt des Bogens
;
; Autor: Frank Riemenschneider
;
; Eingabe:      AX = x-Koordinate Ellipsenmittelpunkt (0-239)
;               BX = y-Koordinate Ellipsenmittelpunkt (0-63)
;               CX = Radius in x-Richtung (0-255)
;               DX = Radius in y-Richtung (0-255)
;               ES = Anfangswinkel in Grad (0-359°)
;               SI = Endwinkel in Grad (0-360°)
; Ausgabe: keine

DATASEG

; Benötigte Variable
extrn ArcStartX : DWORD
extrn ArcStartY : DWORD
extrn ArcEndX : DWORD
extrn ArcEndY : DWORD

CODESEG

PROC AsmPieSlice

; Register retten
        push ax
        push bx
        push cx
        push dx
        push es
        push si
        push di

        call AsmCurve                   ; Ellipsenbogen zeichnen
        mov cx, [WORD ArcStartX]
                                        ; Koordinaten des Startpunktes des Bogens
        mov dx, [WORD ArcStartY]        ; laden
        call AsmLine                    ; Linie vom Mittelpunkt dorthin ziehen
        mov cx, [WORD ArcEndX]          ; Koordinaten des Endpunktes des Bogens
        mov dx, [WORD ArcEndY]          ; laden
        call AsmLine                    ; Linie vom Mittelpunkt dorthin ziehen
        pop di                          ; Alle Registerinhalte wieder
        pop si                          ; laden
        pop es
        pop dx
```

```
        pop cx
        pop bx
        pop ax
        ret                         ; Back home

ENDP


; SLICE.ASM:    Regelt die Parameterübernahme von Turbo Pascal aus für
;               die ASMPIESLICE-Routine in Turbo Assembler
;
; Autor: Frank Riemenschneider
;
; Eingabe:      Parameter der SLICE-Funktion auf dem Stack
;               Slice( XM, YM, XR, YR, AnfWinkel, EndWinkel:WORD );
; Ausgabe:      keine

CODESEG

PROC Slice FAR XMP:WORD, YMP:WORD, XRA:WORD, YRA:WORD, AnfWinkel:WORD, EndWinkel:WORD
        PUBLIC Slice

        mov ax, [XMP]               ; Mittelpunkt-Koordinaten
        mov bx, [YMP]
        mov cx, [XRA]               ; Radius in x- und y-Richtung
        mov dx, [YRA]
        mov es, [AnfWinkel]         ; Anfangs- und Endwinkel
        mov si, [EndWinkel]         ; Koordinaten prüfen
        cmp ax, 0                   ; x-Koordinate < 0?
        jl PieSliceExit
        cmp ax, 239                 ; x-Koordinate > 239?
        jg PieSliceExit
        cmp bx, 0                   ; y-Koordinate < 0?
        jl PieSliceExit
        cmp bx, 63                  ; y-Koordinate > 63?
        jg PieSliceExit
        cmp cx, 0                   ; x-Radius < 0?
        jl PieSliceExit
        cmp cx, 255                 ; x-Radius > 255?
        jg PieSliceExit
        cmp dx, 0                   ; y-Radius < 0?
        jl PieSliceExit
        cmp dx, 255                 ; y-Radius > 255?
        jg PieSliceExit
        push ax
        mov ax, es
        cmp ax, 0                   ; Anfangswinkel < 0°?
        jl PieSliceExit
        cmp ax, 359                 ; Anfangswinkel > 359°?
        jg PieSliceExit
        pop ax
        cmp si, 0                   ; Endwinkel < 0°?
```

```
        jl PieSliceExit
        cmp si, 360                 ; Endwinkel > 360°?
        jg PieSliceExit
        call AsmPieSlice            ; PieSlice-Routine aufrufen
PieSliceExit:
        ret                         ; Back home

ENDP
```

2.5.9 Beliebige Flächen ausfüllen

Der komplizierteste Teil der Grafikroutinen besteht darin, beliebige Flächen auszufüllen. Ich möchte Ihnen an dieser Stelle einen linienorientierten Algorithmus vorstellen, der anders als viele andere Verfahren sich nicht »Punkt für Punkt« in alle Richtungen vortastet, sondern jeweils eine horizontale Linie zeichnet. Bedingt durch die Tatsache, daß die Routine zum Zeichnen einer solchen Linie sehr schnell ist (s.o.), ergeben sich nicht unerhebliche Geschwindigkeitsvorteile gegenüber den punktorientierten Verfahren, zumal wesentlich weniger Rekursionsaufrufe erforderlich sind und damit auch die Gefahr des Stack-Überlaufs vermindert wird.

Als Beispielfläche soll die Figur in Bild 2.7 dienen, wobei der Startpunkt der Füllroutine mit P gekennzeichnet ist. Als erstes wird überprüft, ob dieser Punkt vielleicht schon gesetzt ist, worauf die Routine beendet werden kann. Ist dies nicht der Fall, wird solange um eine Zeile nach oben gewandert, bis ein gesetzter Punkt erreicht wird. In diesem Fall wird wieder um eine Zeile nach unten verzweigt und gleichzeitig ein Schritt nach rechts getätigt. Dieses Verfahren wird solange wiederholt, bis man nicht nur oben gegen eine Begrenzung stößt, sondern auch rechts. Man hat also den rechten oberen Eckpunkt der Figur erreicht. Den Weg dorthin zeigt Bild 2.7, in der Assemblerroutine befinden wir uns an dem Label L2.

Der Eckpunkt wird abgespeichert, da er später noch benötigt wird. Als nächstes wird jetzt solange nach links gewandert, bis auch hier gegen eine Begrenzung gestoßen wird. Nachdem auch diese Koordinaten gespeichert wurden, wird eine Linie zwischen diesen beiden äußeren Punkten gezeichnet. Man sieht, daß sich oberhalb der Linie noch freie Punkte befinden, die bislang nicht erfaßt werden konnten. Deshalb wird in einem nächsten Schritt ab Label L3 eine Zeile nach oben geschritten und vom linken Endpunkt der Linie bis zum rechten Startpunkt geprüft, ob nicht gesetzte Punkte auftreten. Da man sich ja in der Zeile mit der Begrenzung der Figur befindet, kann ein fehlender Punkt nur bedeuten, daß die Figur

nach oben weiterläuft oder die Begrenzung ein Loch hat, was allerdings sehr bitter wäre. Wird also ein fehlender Punkt entdeckt, ruft sich die Routine mit diesem Punkt als Startpunkt rekursiv selbst auf, d.h., der gesamte Algorithmus wird nun auf das freie Feld oberhalb der gezeichneten Linie angewendet. Dafür müssen natürlich zunächst alle Variablen gesichert werden, damit nach Beendigung der Rekursion mit ihnen weitergerechnet werden kann und sie nicht zwischenzeitlich zerstört werden. Nachdem das Feld gefüllt ist, wird nach weiteren Leerpunkten gesucht, bis die rechte Begrenzung der Figur erreicht ist. In unserem Fall ist also ein weiterer Rekursionsaufruf erforderlich. Nachdem die rechte Begrenzung erreicht ist, stellt sich die Figur wie in Bild 2.8 dar.

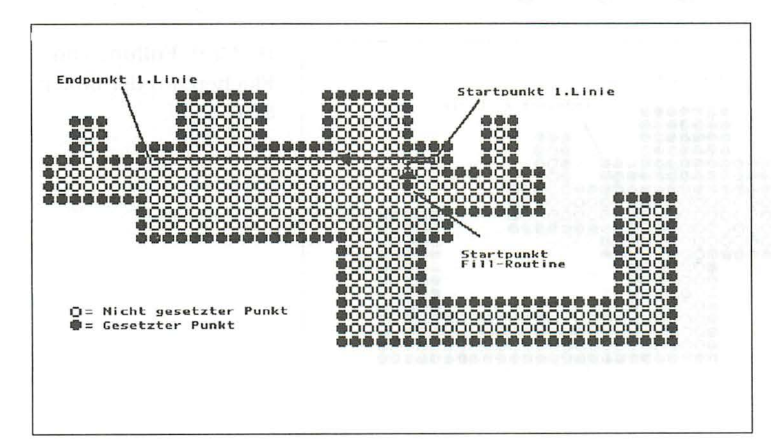


Bild 2.7: Weg zum Startpunkt der Füllung

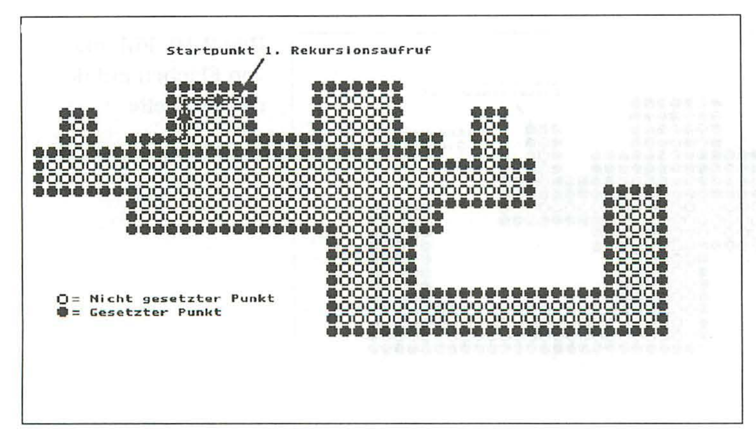


Bild 2.8: Füllung oberhalb der Startlinie

Nun wird die Figur zeilenweise nach unten abgearbeitet (ab Label L5). Dabei wird wiederum die linke sowie rechte Begrenzung gesucht und eine Linie zwischen beiden gezogen. Man muß jedoch aufpassen: Auch nach

unten können sich noch zu füllende Bereiche anschließen. Wie erkennt man diese Fälle?

Den ersten Fall haben wir auf der linken Seite der Figur: Die linke Koordinate der zu zeichnenden Linie ist offenbar kleiner als die der darüberliegenden. Daraus kann man schließen, daß noch ein Flächenteil nach oben anschließt. Man prüft also in der darüberliegenden Zeile (die weiter rechts beginnende), ob noch ein freier Punkt links von diesem Anfangspunkt existiert, und zwar solange, bis man die linke Koordinate der darunterliegenden Linie erreicht hat. In unserem Fall wird ein freier Punkt gefunden und die Routine rekursiv auf diesen freien Flächenteil angewendet (Bild 2.9). Der zugehörige Programmteil befindet sich ab Label L9.

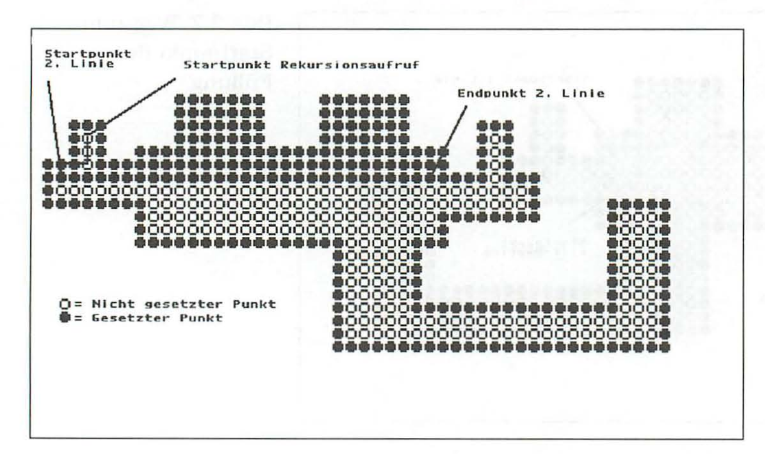


Bild 2.9: Füllung von Flächen auf der linken Seite

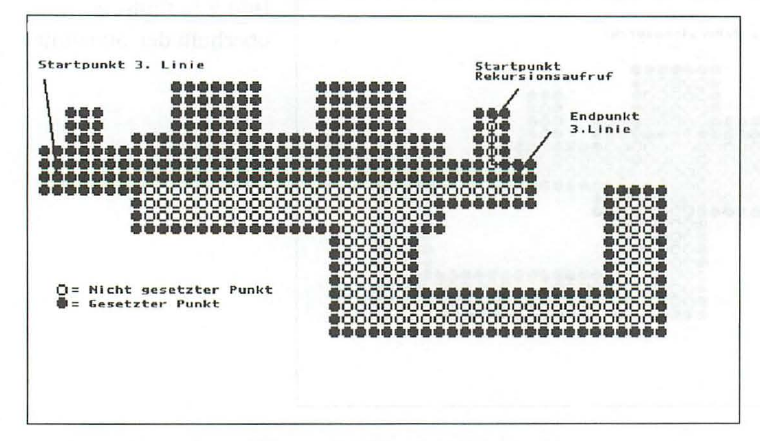


Bild 2.10: Füllung von Flächen auf der rechten Seite

Der zweite Fall tritt ähnlich auf der rechten Seite auf: Hier ist die größere x-Koordinate der unteren Linie ein Indiz dafür, daß sich noch verborgene

Flächenteile oberhalb der unteren Linie befinden könnten. Also wird in Höhe der oberen Linie soweit nach rechts gegangen, bis die rechte Koordinate der unteren längeren Linie erreicht wird. Befindet sich in diesem Teil noch ein freier Punkt, muß die Routine auf diesen angesetzt werden (Bild 2.10). Im Listing wird dieser Fall ab Label L11 behandelt.

Der letzte Fall tritt unten ein: Hier kann es passieren, daß ein Zipfel nach unten herausragt, wie es in unserer Figur der Fall ist. Diesen Fall kann man daran erkennen, daß die untere Linie nicht einen weiter rechts liegenden Endpunkt aufweist als die obere Linie, wie dies im letzten Fall auftrat, sondern einen weiter links liegenden. Man prüft also Spalte für Spalte, ob ein Punkt fehlt, bis man schließlich den rechten Endpunkt der darüberliegenden Linie erreicht hat. Fehlt ein Punkt, wird die Routine wieder rekursiv aufgerufen (Bild 2.11). Ab Label L10 steht der zugehörige Programmteil.

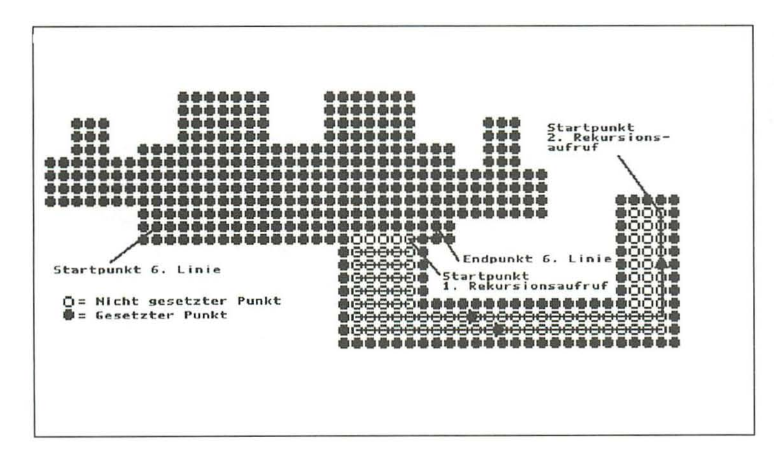


Bild 2.11: Füllung von Flächen auf der unteren Seite

Was aber passiert nun, wenn wie in unserem Fall der untere Zipfel selbst noch einen weiteren nach oben aufweist? Nun, die Routine wird ganz einfach nochmals rekursiv aufgerufen, diesmal jedoch eine Ebene tiefer. Die zulässige Verschachtelungstiefe wird nur durch die Größe des Stack bestimmt. Wenn man aber eine maximale Verschachtelungstiefe von 10 ansetzt, die kaum von einer Figur überboten werden dürfte (Sie können sich ja mal eine Figur ausdenken, die 20 Rekursionen erfordert!), werden jedesmal nur elf Wörter plus der Rücksprungsadresse auf den Stack geschrieben, also 24x10=240 Byte. Dies ist wirklich nicht viel, wenn man bedenkt, daß Turbo Pascal die Stackgröße auf 16384 Byte voreinstellt!

Auf diese Weise wird Zeile für Zeile nach unten abgearbeitet, bis auf einen gesetzten Punkt gestoßen wird. Dann wird solange nach rechts gelaufen,

bis entweder ein leerer Punkt gefunden wird oder die rechte Koordinate der darüberliegenden Zeile erreicht wurde. In letzterem Fall ist die Gesamtfüllung abgeschlossen.

```
; ASMFILL.ASM: Füllt eine beliebige Fläche aus
;
; Autor: Frank Riemenschneider
;
; Eingabe:      AX = x-Koordinate Füllpunkt (0-239)
;               BX = y-Koordinate Füllpunkt (0-63)
; Ausgabe: keine

DATASEG

; Benötigte Variable
speicher21          dw ?
speicher22          dw ?
speicher31          dw ?
speicher32          dw ?
speicher41          dw ?
speicher42          dw ?
tcolor              dw ?
extrn farbe : word;
extrn fillcolor : word

CODESEG

PROC ZieheLinie Near
     push di
     push si
     push dx
     push cx
     mov dx,bx
     Call AsmLine
     pop cx
     pop dx
     pop si
     pop di
     ret
ENDP

PROC TestPunkt Near
     push cx
     push dx
     push di
     push si
     Call AsmTest
     mov al,dl
     pop si
     pop di
     pop dx
     pop cx
```

```
    cmp al,[BYTE tcolor]
    ret
ENDP

PROC AsmFill
    mov dx,ax              ;aktuelle Spalte
    mov bp,bx              ;aktuelle Zeile
    call TestPunkt         ;Startpunkt gesetzt?
    jnz L1
    jmp La1                ;Ja, dann Ende
L1:
    dec bp                 ;Zeile vermindern
    mov ax,dx
    mov bx,bp
    call TestPunkt         ;Punkt gesetzt?
    jnz L1                 ;Nein, weiter nach oben wandern
    inc bp                 ;Ja, ein Schritt nach rechts und
    inc dx                 ;nach unten machen
    mov ax, dx
    mov bx, bp             ;Dieser Punkt auch gesetzt?
    call TestPunkt
    jnz L1                 ;Nein, weiter nach oben wandern

    mov ax,dx              ;Ja, rechte obere Ecke der Figur
    dec ax                 ;erreicht
    mov [speicher21],ax    ;Diese Koordinaten merken
    mov [speicher22],bp
L2:
    dec dx                 ;Solange nach links wandern, bis
    mov ax,dx              ;gesetzter Punkt erreicht wird
    mov bx,bp
    call TestPunkt
    jnz L2

    inc dx                 ;Koordinaten merken (in Si X und in Di Y)
    mov ax,dx
    mov si,ax
    mov bx,bp
    mov di,bx              ;Linie von diesem Punkt zum rechten Punkt
    mov cx, [speicher21]   ;zeichnen
    call ZieheLinie
    dec bp                 ;Eine Zeile nach oben wandern (auf obere
    mov dx,si              ;Begrenzung der Figur)
L3:
    cmp dx,[speicher21]    ;rechte Begrenzung erreicht, dann
    ja L5                  ;keine Lücke mehr nach oben
    mov ax,dx              ;Punkt gesetzt?
    mov bx,bp
    call TestPunkt         ;Nein, d.h. Lücke nach oben
    jnz La40
    inc dx                 ;nächste Spalte bearbeiten
    jmp L3

La40:
```

```
        push si
        push di
        mov ax,[speicher21]
        push ax
        mov ax,[speicher22]
        push ax
        mov ax,[speicher31]
        push ax
        mov ax,[speicher32]
        push ax
        mov ax,[speicher41]
        push ax
        mov ax,[speicher42]
        push ax
        mov ax,dx
        push ax
        mov bx,bp
        push bx
        Call AsmFill            ;Fill-Routine aufrufen
        pop bp
        pop dx
        pop ax
        mov [speicher42],ax
        pop ax                  ;Nach Füllung der Lücke Variablen wieder
        mov [speicher41],ax
        pop ax                  ;vom Stack holen
        mov [speicher32],ax
        pop ax
        mov [speicher31],ax
        pop ax
        mov [speicher22],ax
        pop ax
        mov [speicher21],ax
        pop di
        pop si
        jmp L3

L5:
        mov bx,di               ;linken Ausgangspunkt der Zeile
        inc bx                  ;nehmen (X und Y) für weiteres Vorgehen
        mov bp,bx
        mov ax,si
        mov dx,ax
        mov [speicher32],bx     ;Zeile merken
        mov [speicher42],bx
        call TestPunkt          ;Punkt gesetzt?
        jnz La4                 ;Nein
L6:
        inc dx                  ;Ja : Spalte solange erhöhen, bis
        mov ax,dx
        mov bx,bp
        Call TestPunkt          ;entweder gesetzter Punkt gefunden wurde
        jnz La5
        mov ax,[speicher21]     ;oder Spalte größer als rechte Begrenzung
```

```
        cmp dx,ax
        jna L6                  ;darüberliegenden Zeile ist (d.h. rechte
        ja La5                  ;untere Ecke der Figur gefunden!!!)
La4:
        dec dx                  ;Wenn Ursprungspunkt nicht gesetzt, Spalte
        mov ax,dx
        mov bx,bp               ;solange vermindern, bis gesetzter Punkt
        call TestPunkt
        jnz La4                 ;gefunden wird.
        inc dx
La5:
        mov [speicher31],dx     ;Spalte merken als linken Eckpunkt der Linie
        mov ax,[speicher21]
        cmp dx,ax               ;Falls größer als rechter Eckpunkt der
        jna L8                  ;darüberliegenden Linie, rechte, untere Ecke
        jmp La1                 ;der Figur gefunden, d.h. Fill-Ende
L8:
        inc dx                  ;wenn nicht größer: Rechten Eckpunkt der Linie
        mov ax,dx
        mov bx,bp               ;suchen, indem Spalte erhöht wird
        call TestPunkt
        jnz L8
        dec dx
        mov ax,[speicher31]     ;Linie zeichnen
        mov bx,[speicher32]
        mov cx,dx
        call ZieheLinie
        mov [speicher41],dx     ;Spalte des rechten Eckpunktes der Linie merken
        mov ax,si               ;Beginnt Linie weiter links als die
        dec ax
        cmp ax,[speicher31]     ;darüberliegende???
        ja La22                 ;Ja, Sonderfall 1
        jmp La8
La22:
        mov dx,si               ;Sonderfall 1: Figurzipfel auf linker
        mov bp,di               ;Seite muß gefüllt werden
L9:
        dec dx
        mov ax,dx               ;Freier Punkt gefunden, dann füllen
        mov bx,bp
        call TestPunkt
        jz La81
        push si
        push di
        mov ax,[speicher21]
        push ax
        mov ax,[speicher22]
        push ax
        mov ax,[speicher31]
        push ax
        mov ax,[speicher32]
        push ax
        mov ax,[speicher41]
        push ax
```

```
        mov ax,[speicher42]
        push ax
        mov ax,dx
        push ax
        mov bx,bp
        push bx
        Call AsmFill            ;Fill-Routine aufrufen
        pop bp
        pop dx
        pop ax
        mov [speicher42],ax
        pop ax                  ;Nach Füllung der Lücke Variablen wieder
        mov [speicher41],ax
        pop ax                  ;vom Stack holen
        mov [speicher32],ax
        pop ax
        mov [speicher31],ax
        pop ax
        mov [speicher22],ax
        pop ax
        mov [speicher21],ax
        pop di
        pop si
La81:
        cmp dx,[speicher31]     ;Nur bis zum linken Ausgangspunkt
        jnz L9                  ;der Linie prüfen

La8:
        mov ax,[speicher41]     ;Endet Linie weiter rechts als die
        inc ax                  ;darüberliegende???
        cmp [speicher21],ax
        ja La21                 ;Ja, Sonderfall 2
        jmp La9
La21:
        mov dx,[speicher41]     ;Sonderfall 2: Figurzipfel auf rechter
        mov bp,[speicher42]     ;Seite muß gefüllt werden
L10:
        inc dx
        mov ax,dx               ;Freier Punkt gefunden, dann füllen
        mov bx,bp
        call TestPunkt
        jz La91
        push si
        push di
        mov ax,[speicher21]
        push ax
        mov ax,[speicher22]
        push ax
        mov ax,[speicher31]
        push ax
        mov ax,[speicher32]
        push ax
        mov ax,[speicher41]
        push ax
```

```
        mov ax,[speicher42]
        push ax
        mov ax,dx
        push ax
        mov bx,bp
        push bx
        Call AsmFill            ;Fill-Routine aufrufen
        pop bp
        pop dx
        pop ax
        mov [speicher42],ax
        pop ax                  ;Nach Füllung der Lücke Variablen wieder
        mov [speicher41],ax
        pop ax                  ;vom Stack holen
        mov [speicher32],ax
        pop ax
        mov [speicher31],ax
        pop ax
        mov [speicher22],ax
        pop ax
        mov [speicher21],ax
        pop di
        pop si
La91:
        cmp dx,[speicher21]     ;Nur bis zum rechten Endpunkt der
        jnz L10                 ;Linie prüfen

La9:
        mov ax,[speicher41]     ;Endet Linie weiter links als die
        dec ax                  ;darüberliegende???
        cmp ax,[speicher21]
        ja La20                 ;Ja, Sonderfall 3
        jmp La10
La20:
        mov dx,[speicher21]     ;Sonderfall 3: Figurzipfel an unterer Seite
        mov bp,[speicher22]     ;muß gefüllt werden
L11:
        inc dx
        mov ax,dx               ;Freier Punkt gefunden, dann füllen
        mov bx,bp
        call TestPunkt
        jz La101
        push si
        push di
        mov ax,[speicher21]
        push ax
        mov ax,[speicher22]
        push ax
        mov ax,[speicher31]
        push ax
        mov ax,[speicher32]
        push ax
        mov ax,[speicher41]
        push ax
```

```
        mov ax,[speicher42]
        push ax
        mov ax,dx
        push ax
        mov bx,bp
        push bx
        Call AsmFill              ;Fill-Routine aufrufen
        pop bp
        pop dx
        pop ax
        mov [speicher42],ax
        pop ax                    ;Nach Füllung der Lücke Variablen wieder
        mov [speicher41],ax
        pop ax                    ;vom Stack holen
        mov [speicher32],ax
        pop ax
        mov [speicher31],ax
        pop ax
        mov [speicher22],ax
        pop ax
        mov [speicher21],ax
        pop di
        pop si
La101:
        cmp dx,[speicher41]       ;Nur bis zum linken Ausgangspunkt der
        jnz L11                   ;Linie prüfen

La10:
        mov si,[speicher31]       ;Darüberliegende Linie = aktuelle Linie
        mov di,[speicher32]
        mov ax,[speicher41]
        mov [speicher21],ax
        mov ax,[speicher42]
        mov [speicher22],ax
        jmp L5                    ;nächste Linie bearbeiten

La1:
        ret

ENDP

; FILL.ASM:      Regelt die Parameterübernahme von Turbo Pascal aus für
;                die ASMFILL-Routine in Turbo Assembler
;
; Autor: Frank Riemenschneider
;
; Eingabe:       Parameter der FILL-Funktion auf dem Stack
;                Fill( X, Y, Farbe:WORD );
; Ausgabe:       keine

CODESEG

PROC Fill FAR XMP:WORD, YMP:WORD, FARB:WORD
        PUBLIC Fill
```

```
        mov ax, [FARB]
        mov [tcolor],ax
        mov ax, [XMP]           ; Mittelpunkt-Koordinaten
        mov bx, [YMP]
        cmp ax, 239             ; x-Koordinate > 239?
        jg FillExit
        cmp bx, 63              ; y-Koordinate > 63?
        jg FillExit
        push cx
        push dx
        push si
        push di
        push bp
        mov dx,[farbe]
        push dx
        mov dx,[fillcolor]
        mov [farbe],dx
        call AsmFill
        pop dx
        mov [farbe],dx
        pop bp
        pop di
        pop si
        pop dx
        pop cx
FillExit:
        ret                     ; Back home

ENDP
```

2.5.10 Die Text-Routine

Der Grafikmodus bietet gegenüber dem Textmodus fast nur Vorteile, kann man doch jeden Pixel einzeln ansteuern. Der einzige Nachteil besteht darin, daß man eben keinen Text auf den Bildschirm schreiben kann: Der Videocontroller interpretiert eingegebene Byte-Werte eben nicht mehr als die ASCII-Daten eines Zeichens, sondern als Bitmuster eines Grafikbyte. Daher ist es erforderlich, eine eigene Routine zu programmieren, wenn man Text auch auf dem Grafikschirm darstellen möchte. Man sollte diese Gelegenheit beim Schopf ergreifen und den etwas einfältigen Textmodus des Videocontrollers erweitern: So ist es sicherlich sinnvoll, eine Vergrößerung des Textes in x- und y-Richtung zuzulassen, um Überschriften besser herausheben zu können. Auch das Kippen des Textes um 90, 180,

270 oder 360 Grad kann zur Beschriftung bestimmter Grafiken geeignet sein. Wie kann man dies realisieren?

Nun, zunächst muß man sich darüber im klaren sein, wie ein Zeichen überhaupt aufgebaut ist. Jede moderne Grafikkarte besitzt ein sogenanntes Character-ROM. Hierbei handelt es sich um einen Baustein, in dem die Grafikdaten jedes Zeichens byteweise nach einem bestimmten, von Grafikkarte zu Grafikkarte verschiedenen Muster abgelegt sind. Will der Controller ein Zeichen im Textmodus auf den Bildschirm bringen, sieht er gewissermaßen im ROM nach und liest die entsprechenden Daten aus. Das Problem besteht nun darin, daß man als Programmierer keinen Zugriff auf das Character-ROM hat, da dieses nicht »ausdekodiert« ist, d.h., man hat über den Adreßbus keine Möglichkeit, das ROM zu adressieren. Zum Glück gibt es jedoch beim PC die IBM-Color-Karte, die sogenannte CGA-Karte. Diese Trivialkarte besitzt kein Charakter-ROM. Deshalb implementierte man im BIOS des Computers (ab Segment $F000, Offset $FA6E) die Grafikdaten eines 8x8-Pixel-Zeichensatzes, um überhaupt Zeichen mit dieser Karte darstellen zu können. Dieser Zeichensatz ist also immer verfügbar, egal ob Sie eine Hercules-, EGA- oder VGA-Karte besitzen, und kann einfach ausgelesen werden. Der Portfolio besitzt dieses ROM natürlich nicht, so daß man hier keinen Zugriff hat. Man muß sich die Tabelle der Grafikdaten somit selbst anlegen.

Da jeder Grafikpunkt 1 Bit belegt, sind für die Speicherung eines Zeichens somit 8x8 Bit = 64 Bit = 8 Byte erforderlich. Dabei sind die Daten in der Reihenfolge der ASCII-Codes abgelegt, so daß man z.B. die Anfangsadresse der Grafikdaten des Zeichens »A« (ASCII-Code 65) nach der Formel

Offset = Tabellenstart + 8x65

berechnen kann. In einem Byte ist immer eine komplette Zeile eines Zeichens abgespeichert, so daß z.B. das zweite Byte für die zweite Zeile zuständig ist usw. Das höchstwertigste Bit eines Bytes repräsentiert dabei den am weitesten links liegenden Grafikpunkt, das niederwertigste den am weitesten rechts liegenden. So kann man sich leicht überlegen, wie z.B. die 8 Byte des Zeichens »A« auszusehen haben:

```
Bit      76543210
Zeile 0  ..xx....
Zeile 1  .xxxx...
Zeile 2  xx  xx..
Zeile 3  xxxxxx..
Zeile 4  xx  xx..
Zeile 5  xx  xx..
Zeile 6  xx  xx..
Zeile 7  ........
```

Die unterste Zeile bleibt immer frei, um ein Unterstreichen des Zeichens zu ermöglichen, die beiden rechten Spalten, um den notwendigen Abstand zum nächsten Zeichen zu erhalten. In der Zeile Null sind die Bits 5 und 4 gesetzt, d.h., das erste Byte weist den Wert 32+16 = 48 auf. Das zweite, vierte, fünfte und sechste Byte sind identisch aufgebaut und weisen den Wert 128+64+8+4 = 204 auf. Auf die gleiche Weise können die Werte des ersten und dritten Bytes ermittelt werden, während das siebte Byte Null ist. Wenn man sich beim PC mit DEBUG den entsprechenden Speicherinhalt ab $F000:$FC76 ansieht, erkennt man, daß die ersten acht Byte unser »A« darstellen.

Die Textroutine muß also abhängig davon, ob ein Bit gesetzt ist oder nicht, den entsprechenden Grafikpunkt setzen oder löschen, wobei jeder Punkt in x-Richtung so oft gesetzt werden muß, wie dies durch den Vergrößerungsfaktor festgelegt wurde. Normalerweise erhält man den nächsten Punkt innerhalb einer Zeile dadurch, daß man einen Punkt in x-Richtung fortschreitet und die y-Koordinate unverändert läßt. Um ein Kippen des Textes zu ermöglichen, kann man diesen Weg aber auch flexibel handhaben, so würde z.B. durch eine Bewegung um einen Punkt in y-Richtung und das Konstanthalten der x-Koordinate eine Rotation um 90 Grad erreicht. Allgemein gesprochen benötigt man also zwei Variablen, die festlegen, um wie viele Pixel in x- bzw y-Richtung gelaufen werden muß, um den nächsten Punkt innerhalb einer Zeile zu erreichen. In unserer Routine sind die Variablen *su* (Bewegung nach oben/unten) und *sl* (Bewegung nach links/rechts) dafür zuständig.

Hat man eine Zeile abgearbeitet, muß man das nächste Byte auslesen und an den Anfang der folgenden Zeile springen. Auch hierfür werden zwei Variablen *zu* (Bewegung nach oben/unten) und *zl* (Bewegung nach links/rechts) eingeführt, die der Routine mitteilen, um wie viele Punkte in x- bzw. y-Richtung der neue Zeilenanfang relativ zum Zeilenanfang der vorhergehenden Zeile zu sehen ist. Bei horizontaler Textausrichtung muß man natürlich einen Punkt in y-Richtung laufen, während die x-Koordinate konstant bleibt (Differenz: Null Punkte). Bei gekipptem Text sind entsprechend andere Werte erforderlich. An dieser Stelle fällt auf, daß man praktisch als kostenlose Zugabe die Möglichkeit der Kursivschrift hat, wenn man den neuen Zeilenanfang so festlegt, daß man sich um einen Punkt in x- und y-Richtung bewegt. Insgesamt bieten sich zwölf sinnvolle Schriftvariationen an. Die folgende Tabelle listet den jeweiligen Schrifttyp zusammen mit den erforderlichen Daten für *su*, *sl*, *zu* und *zl* auf:

Schrifttyp	Rotation	zl	zu	sl	su
Normal	0 Grad	0	1	1	0
Linkskursiv	0 Grad	–1	1	1	0
Rechtskursiv	0 Grad	1	1	1	0
Normal	90 Grad	1	0	0	–1
Linkskursiv	90 Grad	1	–1	0	–1
Rechtskursiv	90 Grad	1	1	0	–1
Normal	180 Grad	0	–1	–1	0
Linkskursiv	180 Grad	–1	–1	–1	0
Rechtskursiv	180 Grad	1	–1	–1	0
Normal	270 Grad	–1	0	0	1
Linkskursiv	270 Grad	–1	–1	0	1
Rechtskursiv	270 Grad	–1	1	0	1

Nachdem ein Zeichen auf den Bildschirm gebracht wurde, kann mit dem nächsten Zeichen fortgefahren werden. Der Textroutine wird ein Pointer (Segment/Offset) auf den auszugebenden String übergeben, wobei die Länge des Strings im ersten Zeichen auszulesen ist.

```
; ASMTEXT.ASM: Schreibt einen Text in den Grafik-Bildschirm
;
; Autor: Frank Riemenschneider
;
; Eingabe:      AX = x-Koordinate (0-239)
;               BX = y-Koordinate (0-63)
;               SI = Offsetadresse des Textes
; Ausgabe:      keine

DATASEG

CharSetSeg = 0b000h
CharSetOfs= 007d0h
extrn  Farbe : word
extrn zu : word
extrn zl : word
extrn su : word
extrn sl : word
extrn zch : word
extrn spt : word
extrn xver : word;
extrn yver : word;

CODESEG
```

```
PROC AsmText

        push ax             ; Erst mal alle Register retten, die
        push bx             ; nicht schnell genug in den nächsten
        push cx             ; Stack kommen
        push dx
        push es
        push si
        push di

        mov dx, CharSetSeg  ; Segment der Zeichentabelle in
        mov es, dx          ; ES laden
        push ds
        mov cx, [TextSeg]   ; Segment Text holen
        mov ds,cx
        mov cl, [si]        ; Länge des Textes holen
        xor ch, ch
        pop ds
        inc si              ; Zeiger auf ersten Buchstaben

TextLoop1:
        push ax
        push bx
        push cx             ; Zeichenzähler retten
        push ds             ; Datensegment Pascal retten
        mov cx, [TextSeg]   ; Segment Text holen
        mov ds,cx
        mov cl, [si]        ; ASCII-Code des Buchstabens holen
        pop ds
        push bx
        xor ch, ch
        shl cx, 1           ; Zeichenmatrix beginnt innerhalb der
        mov bx, cx
        shl cx, 1           ; Tabelle an der Adresse (ASCII-Code x 6)
        add cx, bx
        pop bx
        mov di, CharSetOfs  ; Offset der Zeichentabelle laden
        add di, cx          ; ES:DI auf erste Zeile der Matrix
        mov cx, 6           ; Spaltenzähler laden

TextLoop2:
        push cx             ; Spaltenzähler retten
        mov cx, [xver]      ; Vergrößerungsfaktor X

TextLoop5:
        mov dl, [es:di]     ; Zeile der Zeichenmatrix holen
        push cx
        push ax             ; Koordinaten merken
        push bx             ; von Ausgangspunkt Spalte
        push di             ; Zeiger auf Zeichentabelle retten
        push es
        mov cx, 8           ; Pixelzähler laden
TextLoop3:
        push cx             ; Pixelzähler retten
```

```
        mov dh,1                ; Annahme : Punkt gesetzt
        shr dl, 1               ; Pixel ins Carry schieben
        jc TextLab1             ; Punkt gesetzt -> in Vordergrundfarbe
                                ; setzen
        mov dh, 0               ; Punkt in Hintergrundfarbe setzen
TextLab1:
        mov cx, [yver]

TextLoop4:
        cmp dh,0
        je TextLab6
        call AsmPlot            ; Punkt setzen
TextLab6:
        add bx, [zu]            ; y-Koordinate weiterzählen
        add ax, [zl]            ; x-Koordinate weiterzählen
        loop TextLoop4

        pop cx                  ; Pixelzähler wieder laden
loop TextLoop3                  ; Weitere Punkte setzen

        pop es                  ; Zeiger auf Zeichentabelle wieder laden
        pop di
        pop bx                  ; y-Koordinate zurücksetzen
        pop ax                  ; x-Koordinate zurücksetzen
        add ax, [sl]            ; Nächste Spalte bearbeiten
        add bx, [su]

        pop cx
        loop TextLoop5;

        pop cx                  ; Spaltenzähler wieder laden
        inc di
        loop TextLoop2

        pop cx                  ; Zeichenzähler wieder laden
        pop bx
        pop ax
        add ax, [zch]           ; Nächstes Zeichen
        add bx, [spt]
        inc si                  ; plotten
        loop TextLoop1

        pop di                  ; Alle Register wieder von den Bäumen,
        pop si                  ; äh, vom Stack herunterholen
        pop es
        pop dx
        pop cx
        pop bx
        pop ax
        ret                     ; Back home

ENDP
```

```
; TEXT.ASM:     Regelt die Parameterübernahme von Turbo Pascal aus für
;               die ASMTEXT-Routine in Turbo Assembler
;
; Autor: Frank Riemenschneider
;
; Eingabe:      Parameter der TEXT-Prozedur auf dem Stack
;               TestPixel( X, Y, TextSeg, TextOfs : WORD );
; Ausgabe:      keine

CODESEG

PROC Text FAR Xkoord:WORD, Ykoord:WORD, TextSeg:WORD, TextOfs:WORD
        PUBLIC Text

        mov ax, [TextOfs]
        mov si, ax
        mov ax, [Xkoord]        ; x- und y-Koordinaten in die
        mov bx, [Ykoord]        ; entsprechenden Register bringen
        call AsmText            ; Text-Routine aufrufen
TextExit:
        ret                     ; Back home

ENDP
```

Zum Abschluß möchte ich Ihnen noch die Struktur der Grafik-Unit an Hand der unterschiedlichen Abhängigkeiten und Aufrufe aller Grafikroutinen nahebringen:

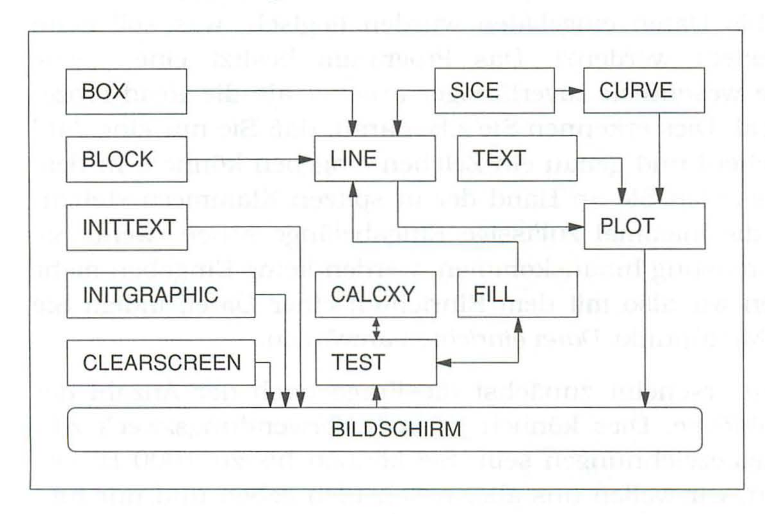


Bild 2.12: Strukturierung der Grafikbefehle

2.6 Chart-Grafiken auf dem Portfolio

Nachdem die theoretischen Grundlagen für die Grafikprogrammierung nun vorhanden sind, wollen wir uns natürlich auch praktische Ergebnisse anschauen. Als Demonstrationsobjekt habe ich mir die Darstellungen von Chart-Grafiken ausgedacht, da es sich hierbei um ein Thema handelt, für das der Portfolio von allen Computern am ungeeignesten zu sein scheint. Wie Sie aber gleich sehen werden, kann man den Werbeslogan einer Autofirma (»Nichts ist unmöglich!«) durchaus auch für den Portfolio mißbrauchen. Die Bilder in diesem Abschnitt wurden übrigens von der PC-Version dieses Programms übernommen, da sie doch wesentlich attraktiver aussehen als die des Mini-Bildschirms des Portfolio.

Zunächst soll es darum gehen, Ihnen die Funktion des Beispielprogramms nahezubringen. Starten Sie es dazu bitte mit dem Befehl:

CHART.EXE

Es handelt sich um ein Chart-Grafik-Programm, dessen Aufgabe darin besteht, z.B. Umsatzzahlen oder Wahlergebnisse grafisch aufzubereiten und damit verständlicher zu machen. Nach dem Programmstart können Sie mit [F1] ins Hauptmenü gelangen. Von dort können Sie in das *Datenmenü* oder *Grafikmenü* weiterverzweigen. Das Grafik-Menü bleibt dabei so lange unwählbar, bis Daten eingeladen wurden (logisch, was soll denn ohne Daten dargestellt werden?). Das Programm besitzt eine eigene Eingaberoutine, die wesentlich zuverlässiger arbeitet als die Read-Prozedur von Turbo Pascal. Dies erkennen Sie z.B. daran, daß Sie nur eine Zahl (keine anderen Zeichen) und genau ein Zeichen eingeben können. In dem ganzen Programm werden Sie an Hand der in spitzen Klammern stehenden Eingabefelder die maximal zulässige Eingabelänge sehen, wenn Sie über die rechte Begrenzung hinauskommen, werden keine Eingaben mehr akzeptiert. Beginnen wir also mit dem Einrichten einer Datei, indem Sie im Dateimenü den Menüpunkt *Datei einrichten* anwählen.

Auf dem Bildschirm erscheint zunächst die Frage nach der Anzahl der sogenannten Datenblöcke. Dies können je nach Verwendungszweck z.B. Parteien oder Artikelbezeichnungen sein. Sie können bis zu 1000 Blöcke in der Datei anlegen, wir wollen uns aber bescheiden geben und nur fünf Blöcke nehmen. Nun werden Sie für jeden Datenblock nach einem Titel gefragt; dies können z.B. Abkürzungen für Parteinamen (CDU, CSU, SPD, Grüne etc.) oder Artikelbezeichnungen sein (Omo, Sunil etc. bei einem Waschmittelhändler). Anschließend werden Sie nach dem Laufwerk und

dem Pfadnamen gefragt, auf dem die Datei angelegt werden soll. Geben Sie hier `a:\pfadname` für ein CCM an oder `c:\pfadname`, falls Sie die eingebaute RAM-Disk benutzen möchten. Soll die Datei ins Hauptverzeichnis geschrieben werden, kann die Eingabe des Pfadnamens natürlich entfallen, wobei aber zu beachten ist, daß hinter der Laufwerksbezeichnung kein Backslash (\) eingegeben werden darf, da die Datei sonst ins aktuelle Verzeichnis geschrieben wird. Nach der Eingabe stellt das Programm fest, wieviel Speicherplatz auf dem Laufwerk vorhanden ist und berechnet damit die maximal zulässige Anzahl der sogenannten Datensätze. Dies können z.B. bei Parteien die zugehörigen Wahlen sein (1972, 1976 etc.) oder bei Artikeln die monatlichen Umsatzzahlen. Die Dateigröße wird wirklich nur durch den zur Verfügung stehenden Speicherplatz des Laufwerks begrenzt, so daß Sie bei der PC-Version des Programms bei der Verwendung einer Festplatte einen gigantischen Wert erhalten. Geben Sie z.B. die Zahl 100 ein. Anschließend muß noch der Dateiname eingegeben werden, der maximal acht Zeichen lang sein darf, wie es DOS fordert. Daraufhin wird die Datei eingerichtet, d.h.; es werden Fantasiedaten auf das Laufwerk geschrieben, die den benötigten Speicher allokieren. Durch dieses Verfahren ist gewährleistet, daß Sie die vorher festgelegten Datenzahlen später auch eingeben können, da andere Programme den Speicherplatz nicht mehr »stibitzen« können. Auf der anderen Seite wird später dann nur noch in diese Datei geschrieben, d.h., es werden keine Daten mehr angehängt, so daß die Dateigröße über ihr gesamtes »Leben« konstant bleibt, egal wie viele Datensätze Sie eingeben. Natürlich ist es aber nicht möglich, mehr Eingaben zu tätigen als bei der Einrichtung festgelegt wurde. Falls Sie also noch unsicher sind, wie viele Datensätze Sie einmal benötigen werden, legen Sie lieber eine zu große als zu kleine Anzahl fest. Nachdem die Datei eingerichtet wurde, wird wieder ins Hauptmenü zurückgesprungen.

Als nächstes müssen Daten eingegeben werden. Dazu wählen Sie den Menüpunkt *Daten eingeben* im Datenmenü aus. Nach der Eingabe des Laufwerks und des Dateinamens (die Voreinstellungen können mit [Return] übernommen werden) werden Sie nach dem ersten Datensatz gefragt, der bearbeitet werden soll. Geben Sie hier eine Zahl zwischen 1 und dem letztmöglichen Datensatz ein (hier: 100), sinnvoll ist die 1, da ja noch überhaupt keine Daten eingegeben wurden. Nun erscheint der ausgewählte erste Datensatz samt Fantasietitel (FMRIE) sowie ein Mini-Untermenü mit folgenden Funktionen: Durch [E] wird der aktuell angezeigte Datensatz (hier: 1) bearbeitet. Konkret heißt dies, daß nacheinander alle Datenblöcke mit den zugehörigen Werten (Fantasiewerte 3006.65, da

noch keine sinnvollen Daten eingegeben wurden) angezeigt werden und eingegeben werden können. Hier zeigt sich nun eine weitere Funktion unserer Luxus-Eingaberoutine: Sie können den Wert durch den Druck der [Return]-Taste übernehmen oder eine neue Eingabe starten. In diesem Fall verschwindet die Voreinstellung aus dem Eingabefeld und braucht somit nicht komplett überschrieben zu werden. Geben Sie nun irgendwelche Zahlen für die einzelnen Datenblöcke ein. Zum Schluß können Sie noch einen Untertitel für den Datensatz festlegen, z.B. LW90 (Landtagswahl 1990) bei Wahlen oder Jan89 (Januar 1989) bei Umsatzzahlen. Der Datensatz wird nun in die Datei eingetragen. Mit den Funktionen [N] und [V] springen Sie zu dem vorhergehenden bzw. nächsten Datensatz, mit [+] und [-] wird in Zehnerschritten gesprungen. Geben Sie ein paar Datensätze zur Übung ein und verlassen Sie den Menüpunkt, indem Sie [A] betätigen.

Als nächsten Dateimenüpunkt haben wir das Einlesen von Daten. Nach der Anwahl des Menüpunktes *Daten laden* müssen Sie erneut Laufwerk und Dateinamen eingeben. Weil auf der Diskette zum Buch eine sehr umfangreiche Beispieldatei vorhanden ist, geben Sie bitte als Laufwerk `a:` und als Dateinamen `demo` ein. In dieser Datei sind Getränkeumsätze von 30 Marken im Zeitraum von 1906 bis 1989 gespeichert. Auf dem Bildschirm können Sie nun 17 der 30 Datenblöcke auswählen, die in den Speicher eingeladen werden sollen. Dazu werden die zugehörigen Titel aufgelistet. Mit den Cursor-Tasten [↓] und [↑] können alle Artikel durchgerollt werden, durch die Tasten [Bild↓] und [Bild↑] wird bildschirmweise gesprungen. Auf der rechten Bildschirmseite befindet sich ein sogenanntes *Gadget* (Dingsda). Dieser Rollbalken zeigt die aktuelle Bildschirmposition an, was besonders sinnvoll ist, wenn Sie mehrere hundert Datenblöcke zu verwalten hätten. Falls Sie einen Datenblock laden möchten, bewegen Sie den Auswahlpfeil vor dessen Namen und drücken Sie die Leertaste. Es erscheint ein Stern vor dem jeweiligen Namen, um die Auswahl zu quittieren. Durch einen erneuten Druck der Leertaste wird der Datenblock wieder deselektiert, der Stern verschwindet. Haben Sie bereits 17 Datenblöcke markiert, können Sie keinen weiteren mehr festlegen, da 17 die voreingestellte maximal zulässige Anzahl der Datenblöcke darstellt. Sie müssen dann erst wieder einen oder mehrere Blöcke deselektieren, um weitere anwählen zu können. Durch [Return] wird die Auswahl abgeschlossen. Als nächstes müssen Sie den Datensatz festlegen, an dem die Auswahl der Datensätze gestartet werden soll. Das bedeutet nicht, daß dieser geladen wird, sondern nur, daß der Dateipointer auf diesen Datensatz gesetzt wird. Nun erscheint ein ähnliches Menü wie bei der Dateneingabe.

Mit [E] wird der aktuell angezeigte Datensatz geladen, mit den Tasten [N], [V], [+] und [-] bewegen Sie sich innerhalb der Datei. Zusätzlich haben Sie mit der Taste [R] die Möglichkeit, mehrere Datensätze in einem Rutsch zu laden: Dazu positionieren Sie den Pointer auf dem ersten zu ladenden Datensatz, wählen mit [R] diese Funktion an und geben anschließend die Zahl der zu ladenden Datensätze ein. Maximal können 700 Datensätze geladen werden, so daß insgesamt 17x700 = 11900 Zahlen im Speicher gehalten werden können. Falls Sie alle Datensätze beisammen haben, können Sie den Menüpunkt mit [A] verlassen.

Jetzt können wir endlich zur grafischen Darstellung der Daten übergehen. Als erstes wollen wir uns mit der zweidimensionalen Balkengrafik beschäftigen. Hierbei werden bis zu sechs Datenblöcke und bis zu 36 Datensätze verarbeitet, indem die Zahlen als Balken nebeneinander dargestellt werden. Man kann also sowohl zwischen einzelnen Datenblöcken und Datensätzen vergleichen (z.B. prozentuale Wahlergebnisse, Umsätze etc.). Die diversen Datenblöcke innerhalb eines Datensatzes werden durch unterschiedliche Muster herausgestellt (Bild 2.13).

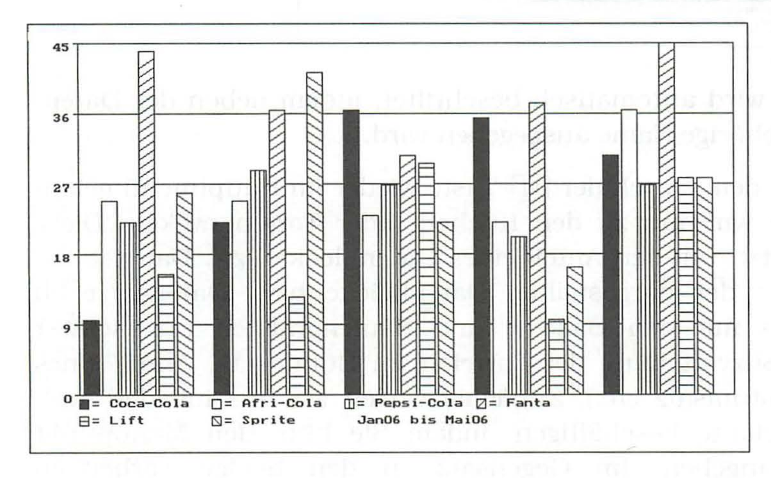


Bild 2.13: Balkengrafik

Nach der Auswahl des Menüpunktes *Balkengrafik h* im Grafikmenü werden Sie zunächst gefragt, welches der erste anzuzeigende Datenblock sein soll und wie viele Datenblöcke Sie anzeigen möchten (indirekt durch Festlegung des letzten Datenblocks). Abhängig von der Anzahl der Blöcke können Sie dann entsprechendes für die Datensätze festlegen. Nun sollte die Grafik auf dem Bildschirm aufgebaut werden. Das Programm beschriftet die Grafik automatisch, indem zu jedem Muster der zugehörige Datenblock geschrieben wird und zudem der erste und letzte Datensatz

erwähnt werden. Durch die [Return]-Taste wird ins Hauptmenü zurückgekehrt.

Als nächstes wollen wir die ganze Sache dreidimensional darstellen. Wählen Sie hierzu den Menüpunkt *Säulengrafik* im Grafikmenü an. Die Säulengrafik entspricht der Balkengrafik, die Säulen der einzelnen Datenblöcke werden jedoch dreidimensional dargestellt (Bild 2.14).

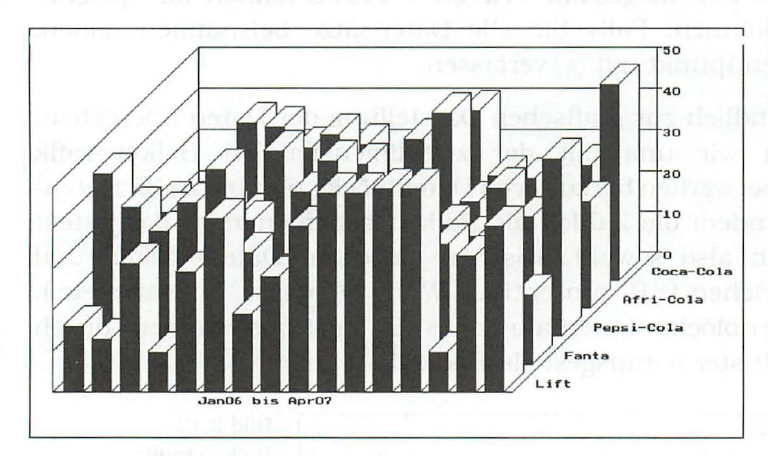


Bild 2.14: Dreidimensionale Säulengrafik

Auch diese Grafik wird automatisch beschriftet, indem neben der Datenblockreihe der zugehörige Name ausgegeben wird.

Nachdem Sie nach dem Druck der [F1]-Taste wieder im Hauptmenü gelandet sind, kommen wir nun zu den Kuchen- oder Tortengrafiken. Diese werden dazu benutzt, um den Anteil eines Datenblocks bzw. Datensatzes an der Gesamtheit der dargestellten Datenblöcke bzw. Datensätze zu verdeutlichen, was mit den Balken- und Säulengrafiken nur schwer möglich ist (z.B. Sitzverteilung der Parteien im Bundestag, Anteil eines Artikels am Gesamtumsatz etc.). Zunächst wollen wir uns mit der zweidimensionalen Variante beschäftigen, indem Sie bitte den Menüpunkt *Tortengrafik 2d* eingeben. Im Gegensatz zu den beiden vorherigen Grafikarten kann hier immer nur ein Datenblock mit mehreren Datensätzen oder ein Datensatz mit mehreren Datenblöcken angezeigt werden. Abhängig davon, ob Sie bei den Datenblöcken nur einen auswählen (indem Sie als ersten und letzten Datenblock die gleiche Nummer angeben), haben Sie die Möglichkeit, einen oder mehrere Datensätze auszuwählen. Zunächst könnten Sie z.B. mehrere Datenblöcke auswählen (bis zu acht erlaubt). Dann werden Sie bei den Datensätzen zwar noch nach dem ersten darzustellenden gefragt, das Programm verhindert aber automatisch eine Eingabe des letzten Datensatzes, da ja nur einer verar-

beitet werden kann. Um nicht einfach eine runde Scheibe auf dem Bildschirm zu haben, wurde die Grafik als sogenannte Explosionsgrafik konzipiert, d.h., es sieht so aus, als ob wie bei einem Urknall alle Tortenstücke von der Mitte aus auseinandergesprengt worden wären. Dies sieht attraktiver aus als eine einfache Scheibe, finden Sie nicht auch?

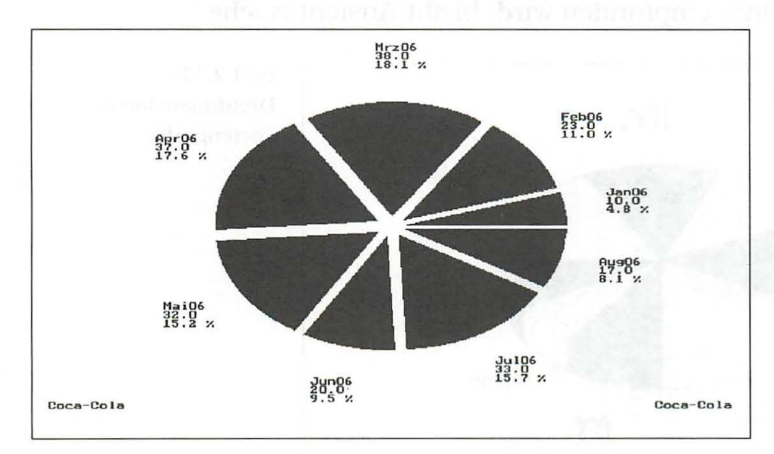


Bild 2.15: Zweidimensionale Tortengrafik mit mehreren Datenblöcken

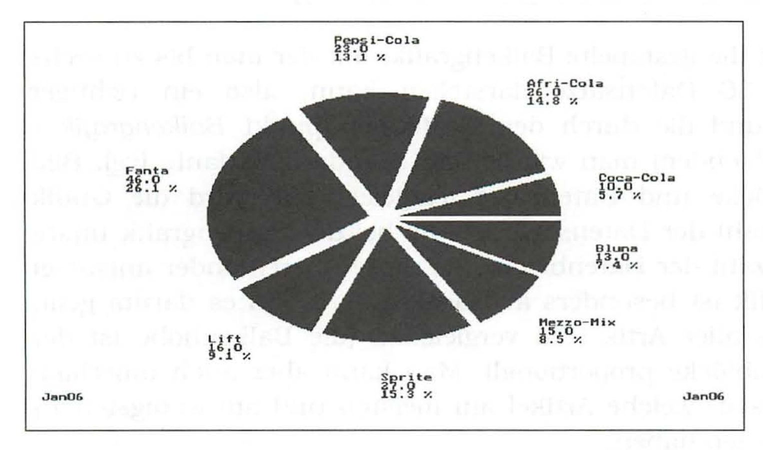


Bild 2.16: Zweidimensionale Tortengrafik mit mehreren Datensätzen

Als Alternative können Sie wie gesagt auch nur einen Datenblock darstellen. In diesem Fall können Sie mehrere Datensätze festlegen, die Frage nach dem letzten Datensatz erscheint also wieder. Wenn Sie Bild 2.15 mit Bild 2.16 vergleichen, stellen Sie fest, daß beide Grafiken völlig unterschiedlichen Aussagewert haben: Während in dem zweiten Bild die Umsätze mehrerer Getränkesorten in einem Monat verglichen werden, um z.B. die Ladenhüter im Winter ausmachen zu können, wird im ersten Bild der Umsatz einer Sorte über mehrere Monate dargestellt, vielleicht um die Werbung in umsatzschwachen Monaten verstärken zu können.

Wenn Sie wieder im Grafikmenü angelangt sind, können Sie durch den Menüpunkt *Tortengrafik 3d* die dreidimensionale Tortengrafik anwählen, für die alles gilt, was auch für die zweidimensionale gesagt wurde. Da aber keine Tortenstücke mehr herausgezogen werden müssen, können Sie statt acht maximal neun Datenblöcke/Sätze grafisch aufbereiten. Welche Grafikform als schöner empfunden wird, bleibt Ansichtssache.

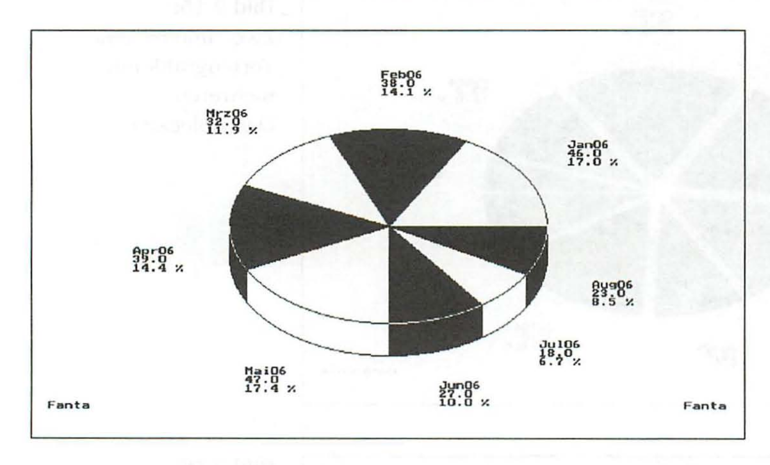


Bild 2.17: Dreidimensionale Tortengrafik

Als letztes verbleibt die gestapelte Balkengrafik, mit der man bis zu sechs Datenblöcke und 36 Datensätze darstellen kann, also ein richtiger »Datenfresser« ist und die durch den Grafikmenüpunkt *Balkengrafik v* ausgewählt wird. Nachdem man wie bei der Standard-Variante (vgl. Bild 2.13) die Datenblöcke und Datensätze festgelegt hat, wird die Grafik aufgebaut. Die Anzahl der Datensätze ist wie bei der Säulengrafik unabhängig von der Anzahl der Datenblöcke, da diese übereinander angezeigt werden. Diese Grafik ist besonders aussagekräftig, wenn es darum geht, den Gesamtumsatz aller Artikel zu vergleichen (die Balkenhöhe ist der Summe aller Datenblöcke proportional). Man kann aber auch innerhalb eines Balkens ablesen, welche Artikel am meisten und am wenigsten zu der Summe beigetragen haben.

Als nächsten Hauptmenüpunkt haben wir eine Toggle-Funktion »Druckstatus«. Falls Sie diesen anwählen, wird die Ausgabe »Ausdruck« auf dem Bildschirm wechselweise zwischen »Ja« und »Nein« wechseln. Bei »Ja« wird nach dem Aufbau einer Grafik eine Hardcopy auf einem Matrixdrucker ausgegeben. Es können alle 9- oder 24-Nadel-Drucker verwendet werden, die Epson-kompatibel sind, d.h., die den ESC/P-Befehlssatz verstehen. Mit dem letzten Hauptmenüpunkt *Programmende* wird das Programm beendet, wobei alle geladenen Daten verlorengehen.

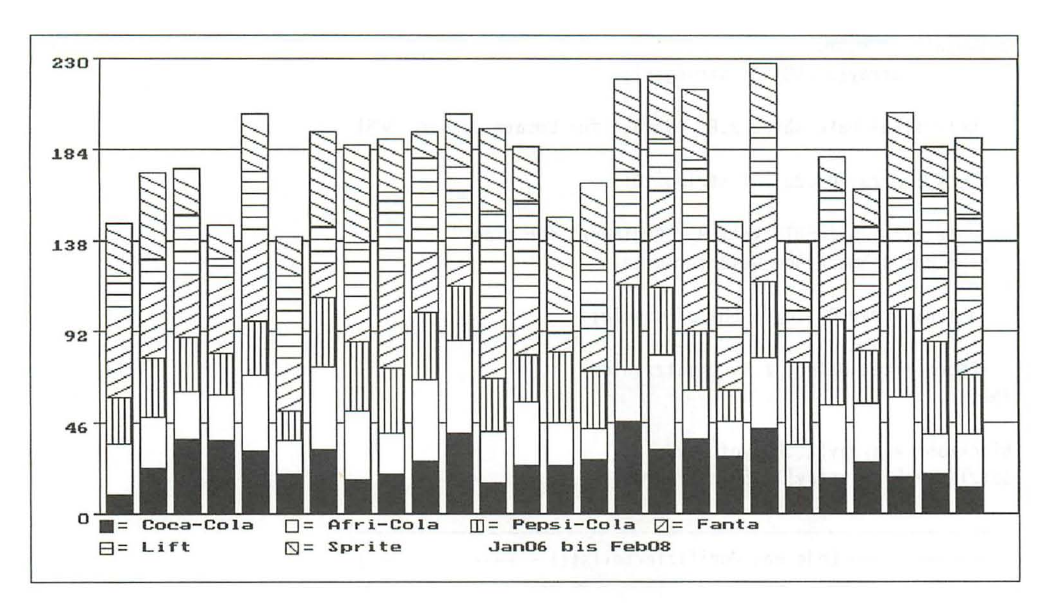


Bild 2.18: Gestapelte Balkengrafik

```
PROGRAM chartgrafik;
{ written 1989,1990 by Frank Riemenschneider
                        Postfach 730309
                        3000 Hannover 71              }

{$I-}
    {Prüfung I/O vom Programm, nicht vom Compiler}

{$N+}
    {Coprozessor 8087 - Befehle erzeugen}
{$E+}
    {Emulator einschalten, weil kein Coprozessor
     installiert ist.}

{----------------------------------------------------------}
{------------ Beginn des Modifizierteils!!! ---------------}
{----------------------------------------------------------}

uses dos,printer,portcrt,portgraf;

{$M 10000,5500,5500}
       {10000 Bytes Stack, Heap  5500 Bytes}

CONST
 maxsaetze : word = 50;          {maximal ladbare Zahl Datensätze}
 maxbloecke : byte = 20;         {maximal ladbare Zahl Datenblöcke}
 graphpath : string = 'c:\';     {Übergabe-Prameter für Grafik-Unit}

TYPE
```

```
datensatz = record
  satz  : array[0..50] of string[5];

  {Untertitel Datensätze z.B. 'Jan85' für Umsatz Januar 1985}

  block : array[0..20] of string[10];

  {Untertitel Datenblöcke z.B. 'Erdinger' für eine
  sehr gut schmeckende Weißbiersorte!}

  zahlen: array[0..20,0..50] of single;

  {Eigentliche Daten, z.B. Umsatzzahlen}
END;

blockzahl = array[1..20] of word;
satzladezahl = array[1..50] of word;

{----------------------------------------------------------------}
{--------------- Ende des Modifizierteils!!! ---------------------}
{----------------------------------------------------------------}

CONST
 maxdatablock : word = 100;     {maximale Blockzahl in Datei}
 hardcopy : array[0..1] of string[4] = ('Ja  ','Nein');
 fullstern : string[39] =
  'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
 randstern : string[1] = 'x';
 datenmenue : array[1..3] of string = ('Datei einrichten','Daten laden',
                                       'Daten eingeben');
 grafikmenue : array[1..5] of string = ('Balkengrafik h','Balkengrafik v',
                                        'Säulengrafik','Tortengrafik 2d',
                                        'Tortengrafik 3d');
 hauptmenue : array[1..4] of string = ('Datenmenü','Grafikmenü',
                                       'Druckstatus','Programmende');

TYPE
 blockdata = array[1..100] of string[10];

VAR
 daten : ^datensatz;            {Insgesamt nur 16 (!) globale
                                 Variablen}
 hpunkt,punkt,druck : byte;
 s : string;
 i,j,k,loadblock : word;
 loadsatz : longint;
 eblo,anzbl,esa,anzsa : word;
 letztpfad : string[25];       {letzter eingegebener Pfad}
 letztname : string[8];    {letzter eingegebener Dateiname}
 input : char;

{----------------------------------------------------------------}
{--------------- U N T E R P R O G R A M M E ---------------------}
{----------------------------------------------------------------}
```

```
{—— Texteingabe: Bildschirmeingabe eines Textes          ——}
{—— Eingabe: Stringvariable, Koordinatenpaar              ——}
{——           x,y (0..79,0..24), maximale Länge,          ——}
{——           Status (0=nur numerische Zeichen,1=alle),   ——}
{——           Voreinstellung                              ——}
{—— Ausgabe: String in Variable ss                        ——}

procedure Eingabe(VAR ss : string ; x,y,maxlaenge,
                  numerisch : byte; vs: string);
VAR taste : byte;
laenge : byte;
i : byte;
flag : boolean;
BEGIN
  GotoXY(x,y);
  write(vs);
  GotoXY(x,y);
  ss := '';
  laenge := 0;
  flag := true;
  REPEAT
    REPEAT
    UNTIL Keypressed;
    taste := Ord(ReadKey);                  {ASCII-Code holen}
    IF flag THEN BEGIN
      IF taste <> 13 THEN BEGIN
        FOR i:= 1 to length(vs) DO BEGIN
          write(' ');
        END;
        GotoXY(x,y);
      END
      ELSE BEGIN
        ss := vs;
      END;
    END;
    flag := false;
    CASE taste of

      {Bei numerischen Zeichen (0,..,9, -,+,.) hier beginnen}

      43,46,48..57 : BEGIN
                      IF laenge < maxlaenge THEN BEGIN
                        inc(x);
                        inc(laenge);
                        write(chr(taste));
                        ss := ss + chr(taste);
                      END;
                     END;

      {Bei anderen Zeichen (a,..,z,A,..,Z,\,[,],(,),!,"
       ,$,%,x,Leer) hier beginnen}

      32..47,58..93,97..154 : BEGIN
```

```
                    IF ((laenge < maxlaenge) and
                        (numerisch <>0)) THEN BEGIN
                      inc(x);
                      inc(laenge);
                      write(chr(taste));
                      ss := ss + chr(taste);
                    END;
                  END;

    {Hier beginnen, wenn Backspace gedrückt. (Über Scan-Code)}

         8   : BEGIN
                 IF laenge > 0 THEN BEGIN
                   dec(x);
                   dec(laenge);
                   GotoXY(x,y);             {Letztes Zeichen löschen}
                   write(' ');
                   GotoXY(x,y);
                   ss := copy(ss,1,Length(ss)-1);
                 END;
               END;
    END;

  UNTIL taste = 13;          {Abschluß mit RETURN}
  writeln;
END;

{-- Standardbild: Hintergrundbildschirm aufbauen             --}
{-- Eingabe: Keine                                           --}
{-- Ausgabe: Keine                                           --}

procedure Standardbild;
VAR i : byte;
BEGIN
   ClrScr;
   write('xxxxxxx M & T  C H A R T  V 1.1 xxxxxxx');
   GotoXY(0,0);
   FOR i := 1 to 6 DO BEGIN
    GotoXY(0,i);
     write(randstern);
     GotoXY(38,i);
     write(randstern);
   END;
   GotoXY(0,7);
   write(fullstern);
END;

{-- Fehler: Fehlermeldung ausgeben                           --}
{-- Eingabe: Meldung (String)                                 --}
{-- Ausgabe: Keine                                           --}

procedure fehler(s : string);
BEGIN
   Standardbild;
```

```
   GotoXY(2,2);
   write(s);
   GotoXY(2,4);
   write('Weiter mit <RETURN> .....');
   readln;
END;


{—— Einrichten: Datei einrichten (beliebig groß)          ——}
{—— Eingabe: Keine                                         ——}
{—— Ausgabe: Keine                                        ——}

procedure Einrichten;
VAR bloecke: word;
s,laenge : string;
drive : byte;
platz,maxsatz,saetze : longint;
block : blockdata;
i,j,k : longint;
f : file;
code : integer;
CONST
dummyzahl : single = 3006.1965;         {Geburtsdatum des Autors}
dummystring : string[5] = 'FMRIE';      {Initialien desselben   }

BEGIN
  Standardbild;
  GotoXY(2,2);
  write('Datenblöcke (1-',maxdatablock,') : <      >');
  str(maxdatablock,laenge);
  REPEAT
    Eingabe(s,22+length(laenge),2,5,0,'');
    val(s,bloecke,code);
  UNTIL ((bloecke >=1) and (bloecke <=maxdatablock));
  FOR i:= 1 to bloecke DO BEGIN
    GotoXY(2,4);
    write('Datenblock Nummer ',i);
    GotoXY(2,5);
    write('Untertitel : <            >');
    Eingabe(s,16,5,10,1,'');
    block[i] := s;
  END;
  Standardbild;
  GotoXY(2,2);
  write('Laufwerk und Pfadname :');
  GotoXY(2,3);
  write('<                          >');
  Eingabe(s,3,3,25,1,letztpfad);
  IF copy(s,length(s),1) <> '\' THEN s := s + '\';
  drive := ord(UpCASE(s[1]))-64;
  platz := DiskFree(drive);
  platz := platz - Round(platz/100);                 {1 % Reserve}
  IF platz > ((bloecke*11)+6) THEN BEGIN               {Zeile 1}
```

```
{Anzahl Datensätze berechnen: Vorhandene Speicherkapazität
insgesamt: platz. In der Info-Datei belegt jeder Datenblock
11 Byte (Stringlänge = 10 plus 1 Byte Länge des String).
Weiterhin je 6 Byte für die Anzahl der Datenblöcke und
Datensätze (word+longint = 6 Byte) = 12 Byte. Der Rest
des Speichers steht für die Hauptdatei zur Verfügung und wird
durch die Anzahl der Datenblöcke geteilt, wobei pro Datensatz
noch 6 Bytes für den Untertitel (5 plus 1 Längenbyte)
reserviert werden müssen.}

  letztpfad := s;
  maxsatz := Trunc((platz-bloecke*11-6)/(6+bloecke*4));
  GotoXY(2,5);
  write('Datensätze (1-',maxsatz,') : <          >');
  str(maxsatz,laenge);
  REPEAT
    Eingabe(s,21+length(laenge),5,10,0,'');
    val(s,saetze,code);
  UNTIL ((saetze >=1) and (saetze <=maxsatz));
  Standardbild;
  GotoXY(2,2);
  write('Dateiname : <        >');
  Eingabe(s,15,2,8,1,letztname);
  letztname := s;

  {Ab hier wird die Info-Datei erstellt}

  Assign(f,chr(drive+64)+':'+s+'.inf');
  rewrite(f,1);
  IF IOResult = 0 THEN BEGIN                        {Zeile 2}
    GotoXY(2,4);
    write('Info-Datei');
    blockwrite(f,bloecke,2);   {Anzahl Datenblöcke schreiben}
    k := IOResult;
    IF k = 0 THEN BEGIN
      blockwrite(f,saetze,4);   {Anzahl Datensätze schreiben}
      k := IOResult;
    END;
    i := 1;
    WHILE ((k = 0) and (i<= bloecke)) DO BEGIN
      GotoXY(15,4);
      write('Block ',i);
      blockwrite(f,block[i],11);
      k := IOResult;              {Block-Untertitel schreiben}
      inc(i);
    END;
    close(f);
    IF k=0 THEN BEGIN                               {Zeile 3}

    {Ab hier wird die Hauptdatei erstellt}

      Assign(f,chr(drive+64)+':'+s+'.dat');
      rewrite(f,1);
      k := IOResult;
```

```
        If k = 0 THEN BEGIN                               {Zeile 4}
          GotoXY(2,4);
          write('Haupt-Datei');
          i := 1;
          WHILE ((k = 0) and (i<= saetze)) DO BEGIN
            GotoXY(15,4);
            write('Datensatz : ',i);
            blockwrite(f,dummystring,6);
            k := IOResult;
            inc(i);                     {Dummy-Untertitel schreiben}
            j := 1;
            WHILE ((k = 0) and (j <= bloecke)) DO BEGIN
              str(j,s);
              GotoXY(15,5);
              write('Datenblock :      ');
              GotoXY(31-length(s),5);
              write(j);
              blockwrite(f,dummyzahl,4);
              k := IOResult;
              inc(j);                     {Dummy-Zahlen schreiben}
            END;
          END;
          close(f);
          IF k<>0 THEN fehler('Fehler beim Schreiben der Hauptdatei!');
        END                                           {von Zeile 4}
        ELSE BEGIN
          fehler('Fehler beim Öffnen der Hauptdatei!');
        END;
      END                                             {von Zeile 3}
      ELSE BEGIN
        fehler('Fehler beim Schreiben der Info-Datei!');
      END;
    END                                               {von Zeile 2}
    ELSE BEGIN
      fehler('Fehler beim Öffnen der Info-Datei!');
    END;
  END                                                 {von Zeile 1}
  ELSE BEGIN
    fehler('Kein Platz auf Laufwerk '+s[1]+': !');
  END;
END;

{—— Scroll: Bildschirmausschnitt nach oben/unten rollen   ——}
{—— Eingabe: Interruptnummer, Zeile für Ausgabe,            ——}
{——          nachrückende Ausgabe, Zeile für Pfeil         ——}
{—— Ausgabe: Keine                                         ——}

procedure scroll(Intnr,zeile : byte; block : string;
                 pfeil : byte);
VAR reg : registers;
BEGIN
  reg.ch := 2;          {Zeile linke obere Ecke}
  reg.cl := 3;          {Spalte linke obere Ecke}
  reg.dh := 5;          {Zeile untere rechte Ecke}
```

```
  reg.dl := 20;          {Spalte untere rechte Ecke}
  reg.bh := 7;           {Leerzeilen in Hintergrundfarbe}
  reg.al := 1;           {Anzahl Zeilen, um die gesrollt wird}
  reg.ah := Intnr;       {nach oben = 6, nach unten = 7}
  Intr(16,reg);          {Interrupt hex. 10, Funktion 6/7,
                          in PC INTERN auf Seite 925/926}
  GotoXY(3,zeile);       {Nachrückende Zeile ausgeben}
  write(block);
  GotoXY(2,pfeil);
  write('>');
END;

{—— waehlen: Datenblöcke aus Liste auswählen             ——}
{—— Eingabe:  Anzahl vorhandener Datenblöcke,            ——}
{——           Datenblocknamen, gewählte Blöcke,          ——}
{——           maximale Anzahl wählbare Datenblöcke       ——}
{—— Ausgabe: Anzahl gewählter Datenblöcke                ——}
function waehlen(bloecke : word; block : blockdata;
         VAR blocknummer : blockzahl; maxblock : word) : word;
VAR
zahl,pos,ya :  word;
wahl : byte;
i,j,k : word;
gewaehlt : blockdata;
BEGIN
   FOR i:= 1 to bloecke DO BEGIN  {Alle Blöcke nicht gewählt}
     gewaehlt[i] := ' ';
   END;
   zahl := 0;
   Standardbild;
   FOR i := 2 to 5 DO BEGIN
     GotoXY(34,i);
     write('[  ]');
   END;
   IF bloecke < 4 THEN BEGIN
      k := bloecke;
   END
   ELSE BEGIN
      k := 4;
   END;
   j := 1;                        {erster angezeigter Datenblock}
   pos := 0;                      {Position des Auswahlpfeiles}
   ya := 2;
   FOR i := j to k DO BEGIN       {erste Blöcke ausgeben}
     GotoXY(3,1+i);
     writeln(gewaehlt[i]+block[i]);
   END;
   GotoXY(2,2);
   write('>');
   REPEAT
     GotoXY(35,2+Trunc(j/((bloecke-3)/3)));
     write('<>');
     wahl := Ord(ReadKey);
```

```
{Wenn Block gewählt, dann als nicht gewählt kennzeichnen}

IF wahl=32 THEN BEGIN
  IF gewaehlt[j+pos] = '*' THEN BEGIN
     gewaehlt[j+pos] := ' ';
     dec(zahl);
  END

 {Wenn Block noch nicht ausgewählt, dann auswählen}

  ELSE BEGIN
     IF zahl<maxblock THEN BEGIN
       gewaehlt[j+pos] := '*';
       inc(zahl);
     END;
  END;
  GotoXY(3,ya);
  write(gewaehlt[j+pos]+block[j+pos]);
END;
IF wahl = 0 THEN BEGIN
  wahl := Ord(readKey);
  GotoXY(35,2+Trunc(j/((bloecke-3)/3)));
  write('  ');
  CASE wahl of

  {Hier fortfahren, wenn Cursor nach oben gedrückt wurde}

    $48 : BEGIN
            IF pos > 0 THEN BEGIN
              GotoXY(2,ya);
              write(' ');
              dec(ya);                {kein Scrollen, nur}
              dec(pos);               {Auswahlpfeil wandert}
              GotoXY(2,ya);
              write('>');
            END
            ELSE BEGIN                {Scrollen erforderlich}
              IF j>1 THEN BEGIN
                scroll(7,2,gewaehlt[j-1]+block[j-1],ya);
                dec(j);
              END;
            END;
          END;

    {Hier fortfahren, wenn Cursor nach unten gedrückt wurde}

    $50 : BEGIN
            IF pos < k-1 THEN BEGIN
              GotoXY(2,ya);
              write(' ');
              inc(ya);                {Kein Scrollen, nur}
              inc(pos);               {Auswahlpfeil wandert}
              GotoXY(2,ya);
              write('>');
```

```
            END
            ELSE BEGIN                        {Scrollen erforderlich}
              IF j+3<bloecke THEN BEGIN
                scroll(6,5,gewaehlt[j+4]+block[j+4],ya);
                inc(j);
              END;
            END;
           END;

        {Hier fortfahren, wenn Bild oben gedrückt wurde}

        $49 : BEGIN
            IF j > 4 THEN BEGIN
              j := j-4;
              FOR i := j to j+3 DO BEGIN      {erste Blöcke ausgeben}
                GotoXY(3,2+i-j);
                writeln(gewaehlt[i]+block[i]+'           ');
              END;
              GotoXY(2,ya);
              write('>');
            END;
           END;

        {Hier fortfahren, wenn Bild unten gedrückt wurde}

        $51 : BEGIN
             IF j+3 < bloecke-3 THEN BEGIN
               j := j+4;
               FOR i := j to j+3 DO BEGIN     {erste Blöcke ausgeben}
                  GotoXY(3,2+i-j);
                  writeln(gewaehlt[i]+block[i]+'           ');
               END;
               GotoXY(2,ya);
               write('>');
             END;
          END;

        END;
      END;

    UNTIL wahl = 13;                          {RETURN schließt Auswahl ab}

   {Hier werden die ausgewählten Blocknummern festgelegt}

   IF zahl>0 THEN BEGIN
     j := 1;
     FOR i:= 1 to bloecke DO BEGIN
       IF gewaehlt[i] = '*' THEN BEGIN
         blocknummer[j] := i;
         daten^.block[j] := block[i];
         inc(j);
       END;
     END;
   END;
```

```
 Standardbild;
 waehlen := zahl;
END;

{—— Kopf: Miniatur-Menü für Procedur 'Eingeben' aufbauen ——}
{—— Eingabe: Keine                                       ——}
{—— Ausgabe: Keine                                       ——}

procedure Kopf(flag : boolean);
CONST s : string[16] = '<N> <V> <+> <-> ';
BEGIN
  Standardbild;
  GotoXY(2,1);
  CASE flag of
    false : write(s+'<E> <A>');
    true  : write(s+'<E> <R> <A>');
  END;
END;

{—— Leerweg: Entfernen von Leerzeichen an Stringanfängen ——}
{—— Eingabe: String mit Leerzeichen                      ——}
{—— Ausgabe: String ohne Leerzeichen                     ——}

function Leerweg(s:string) : string;
BEGIN
  WHILE Copy(s,1,1) = ' ' DO BEGIN
    s := Copy(s,2,Length(s)-1);
  END;
  Leerweg := s;
END;

{—— Einladen: Daten eingeben/Daten laden                ——}
{—— Eingabe: Flag eingeben=false,laden=true             ——}
{—— Ausgabe: Keine                                      ——}

procedure Einladen(flag : boolean);
VAR bloecke,anzahl,IO : word;
s,laenge,ss : string;
s1 : char;
saetze,erster,letzter : longint;
block : blockdata;
blocknummer : blockzahl;
geladen : satzladezahl;
i,j,k : longint;
f : file;
code : integer;
dummystring : string[5];
dummyzahl : single;
taste : byte;

BEGIN
    FOR i:= 1 to maxsaetze DO BEGIN
      geladen[i] := 0;
```

```
END;
Standardbild;
GotoXY(2,2);
write('Laufwerk und Pfadname :');
GotoXY(2,3);
write('<                         >');
Eingabe(s,3,3,25,1,letztpfad);
IF copy(s,length(s),1) <> '\' THEN s := s + '\';
letztpfad := s;
GotoXY(2,5);
write('Dateiname : <        >');
Eingabe(ss,15,5,8,1,letztname);
letztname := ss;
Assign(f,s+ss+'.inf');
reset(f,1);
IF IOResult = 0 THEN BEGIN                          {Zeile 1}

{Info-Datei auslesen}

  blockread(f,bloecke,2);
  k := IOResult;
  IF k=0 THEN BEGIN
    blockread(f,saetze,4);
    k := IOResult;
  END;
  i := 1;
  WHILE ((k=0) and (i <= bloecke)) DO BEGIN
    blockread(f,block[i],11);
    k := IOResult;
    inc(i);
  END;
  close(f);
  Assign(f,s+ss+'.dat');
  IF k = 0 THEN BEGIN                               {Zeile 2}
    loadblock := 0;
    IF flag THEN loadblock := waehlen(bloecke,block,
                              blocknummer,maxbloecke);
    Standardbild;
    GotoXY(2,2);
    write('1. Datensatz (1-',saetze,') :');
    str(saetze,laenge);
    GotoXY(22+length(laenge),2);
    write('<         >');
    REPEAT
      Eingabe(s,23+length(laenge),2,9,0,'1');
      val(s,erster,code);
    UNTIL ((erster >=1) and (erster <=saetze));
    reset(f,1);
    If IOResult = 0 THEN BEGIN
      loadsatz := 0;
      i := erster;
      Kopf(flag);
      REPEAT
```

```
{Position für Anfang eines bestimmten Datensatzes der
 Datei berechnen: Pro Datensatz 6 Byte für Untertitel
 und je 4 Byte pro Wert jedes Datenblocks. Also : Pro
 Datensatz existieren (6+4*Blöcke) Byte.}

        seek(f,(6+4*bloecke)*(i-1));
        blockread(f,dummystring,6);
        IO := IOResult;
        IF IO = 0 THEN BEGIN
          IF flag THEN BEGIN
            FOR k:= 1 to loadsatz DO BEGIN
              IF geladen[k] = i THEN dummystring := 'Load';
            END;
            GotoXY(2,2);
            write('Ladbare Datensätze :           ');
            GotoXY(25,2);
            write(maxsaetze-loadsatz);
          END;
          GotoXY(2,3);
          write('Datensatz ',i,' : ',dummystring,'          ');
          taste := Ord(ReadKey);
          CASE UpCASE(chr(taste)) of
            '+' : IF i<(saetze-9) THEN i:= i+10;
            '-' : IF i>10 THEN i:= i-10;
            'N' : IF i<saetze THEN inc(i);
            'V' : IF i>1 THEN dec(i);
            'E' : BEGIN
                    IF not flag THEN BEGIN
                      j := 1;
                      WHILE ((j<=bloecke) and (IO=0)) DO BEGIN

{Position für Anfang eines bestimmten Datenblocks eines
 bestimmten Datensatzes der Datei berechnen: Pro Datensatz
 existieren (6+4xBlöcke) Bytes (s.o.). Zu dem Anfang des
 Datensatzes müssen noch für jeden Datenblock 4 Byte
 hinzugezählt werden (Single-Zahl), sowie 6 Byte für den
 Untertitel.}

                        seek(f,((6+4*bloecke)x(i-1))+
                               (6+(j-1)*4));
                        blockread(f,dummyzahl,4);
                        IO := IOResult;
                        IF IO = 0 THEN BEGIN
                          GotoXY(2,4);
                          write('                                    ');
                          GotoXY(2,4);
                          write(block[j],' = ',dummyzahl:12:4);
                          GotoXY(2,5);
                          write('Neuer Wert :',
                                '<                >');
                          str(dummyzahl :16:4,ss);
                          Eingabe(s,16,5,15,0,Leerweg(ss));
                          val(s,dummyzahl,code);
                          IF s<>'' THEN BEGIN
```

```
            seek(f,((6+4*bloecke)x(i-1))+
                (6+(j-1)*4));
            blockwrite(f,dummyzahl,4);
            IO := IOResult;
         END;
         inc(j);
       END;
     END;                {von WHILE}
     IF IO = 0 THEN BEGIN
       GotoXY(2,4);
       write('                          ');
       GotoXY(2,5);
       write('Neuer Titel : <     >           ');
       Eingabe(s,17,5,5,1,dummystring);
       IF s<>'' THEN BEGIN
         seek(f,(6+4*bloecke)x(i-1));
         blockwrite(f,s,6);
         IO := IOResult;
       END;
       GotoXY(2,4);
       write('                          ');
       GotoXY(2,5);
       write('                          ');
     END;
     IF IO<>0 THEN BEGIN
       Standardbild;
       GotoXY(2,2);
       write('Fehler bei Zugriff auf Hauptdatei!');
       taste := Ord('A');
       readln;
     END;
   END     {von Flag}

   ELSE BEGIN
     IF ((loadsatz < maxsaetze) and
      (dummystring <> 'Load')) THEN BEGIN
       inc(loadsatz);
       geladen[loadsatz] := i;
       j := 1;
       WHILE ((j<=loadblock) and (IO = 0)) DO BEGIN
         seek(f,((6+4*bloecke)*(i-1))
             +(6+((blocknummer[j]-1)*4)));
         blockread(f,daten^.zahlen
                   [j,loadsatz],4);
         IO := IOResult;
         inc(j);
       END;
       IF IO = 0 THEN BEGIN
         seek(f,((6+4*bloecke)*(i-1)));
         blockread(f,daten^.satz[loadsatz],6);
         IO := IOResult;
       END;
       IF IO <>0 THEN BEGIN
          loadblock := 0;
```

```
                GotoXY(2,4);
                write('Fehler bei Zugriff auf Hauptdatei!');
                GotoXY(2,5);
                write('    Weiter mit <RETURN>.');
                taste := Ord('A');
                readln;
             END;
           END;
         END;
        END;
 'R' : IF ((flag) and (loadsatz < maxsaetze)
        and (dummystring <> 'Load')) THEN BEGIN
         GotoXY(2,5);
         letzter := maxsaetze-loadsatz;
         IF (i + letzter-1) > saetze THEN
           letzter := saetze-i+1;
         write('Anzahl (1-',letzter,') :');
         GotoXY(21,5);
         write('<         >');
         REPEAT
           Eingabe(s,22,5,9,1,'');
           val(s,anzahl,code);
         UNTIL ((anzahl >=1) and
               (anzahl <=letzter));
         k := i;
         WHILE ((k <= i+anzahl-1) and (IO=0)) DO BEGIN
           dummystring := '';
           FOR j:= 1 to loadsatz DO BEGIN
             IF geladen[j] = k THEN
               dummystring := 'Load';
           END;
           IF dummystring <> 'Load' THEN BEGIN
             inc(loadsatz);
             GotoXY(22,5);
             write('         ');
             GotoXY(22,5);
             write(anzahl+i-k);
             GotoXY(25,2);
             write('          ');
             GotoXY(25,2);
             write(maxsaetze-loadsatz);
             geladen[loadsatz] := k;
             j := 1;
             WHILE ((j<= loadblock) and (IO = 0)) DO BEGIN
               seek(f,((6+4*bloecke)*(k-1))
                   +(6+((blocknummer[j]-1)*4)));
               blockread(f,daten^.zahlen
                        [j,loadsatz],4);
               IO := IOResult;
               inc(j);
             END;
             IF IO = 0 THEN BEGIN
               seek(f,((6+4*bloecke)*(k-1)));
               blockread(f,daten^.satz[loadsatz],6);
```

```
                    IO := IOresult;
                  END;
                  IF IO <>0 THEN BEGIN
                    loadblock := 0;
                    GotoXY(2,4);
                    write('Fehler beim Lesen aus Hauptdatei!');
                    GotoXY(7,5);
                    write('Weiter mit <RETURN>.       ');
                    taste := Ord('A');
                    readln;
                  END;
                END;
                inc(k);
              END;
              GotoXY(2,5);
              write('                             ');
             END;
        END;
        IF taste=0 THEN taste := Ord(ReadKey);
      END
      ELSE BEGIN
        loadblock := 0;
        fehler('Fehler bei Zugriff auf Hauptdatei!');
        taste := Ord('A');
      END;

     UNTIL (UpCASE(chr(taste)) = 'A');
     IF IO = 0 THEN close(f);
    END
    ELSE BEGIN
     fehler('Fehler beim Öffnen der Hauptdatei!');
    END;
   END                                            {von Zeile 2}
   ELSE BEGIN
    fehler('Fehler beim Lesen der Info-Datei!');
   END;
  END                                             {von Zeile 1}
  ELSE BEGIN
   fehler('Fehler beim Öffnen der Info-Datei!');
  END;
END;

{—— Grafinit: Initialisiert Pixelgrafik                      ——}
{—— Eingabe: Keine                                           ——}
{—— Ausgabe: x-Koordinate Bildschirmmitte                    ——}

function Grafinit : word;
VAR
  GraphDriver, GraphMode : Integer;
BEGIN
 GraphDriver := 4;
 GraphMode := 1;                        {640 x 350 Grafikpunkte}
 Grafinit := 120;
 InitGraph(graphdriver,graphmode,graphpath);
```

```
 SetColor(15);
END;

{—— BeschrTorte: Beschriftung der Tortendiagramme           ——}
{—— Eingabe: Ausgabetext, Zaehler, Tortenstückanteil         ——}
{—— Ausgabe: Keine                                          ——}

procedure beschrtorte(ss : string; zaehler : word; anteil : single);
VAR xm,ym : integer;
s : string;
BEGIN
     ym := 4+16*(zaehler-Trunc(zaehler/4)*4);
     xm := 75*(Trunc(zaehler/4));
     s := Copy(ss,1,10);
     OutTextXY(xm+12,ym-4,s);
     str(zaehler+1,s);
     OutTextXY(xm,ym,s+'=');
     str(Trunc((anteil/3.6)*10+0.5)/10:4:1,s);
     s := Leerweg(s);
     OutTextXY(xm+12,ym+4,s+' %');
END;

{—— Kuchendiagramm: Aufbau der 2-D-Tortengrafik             ——}
{—— Eingabe: erster Datensatz,letzter Datensatz,             ——}
{——          erster Datenblock, letzter Datenblock           ——}
{—— Ausgabe: Keine                                          ——}

procedure Kuchendiagramm(edb,ldb : word; eds,lds : byte);
VAR xmitte : word;
anteil,summe,halbierende : single;
startwinkel,endwinkel,mittelwinkel,xm,ym : Integer;
s : string;
zaehler,i,j,k : word;
CONST
 ymitte : word = 32;       {Mittelpunkt y-Koordinate}
 xradius : word = 35;      {x-Radius EGA-Grafik}
 verh : single = 0.70;     {Verhältnis y-Radius zu x-Radius}
 expl : word = 8;          {Um soviel Punkte werden die
                            Tortenstücke rausgezogen}
BEGIN
 xmitte := Grafinit+75;
 summe := 0;
 FOR i:= edb to ldb DO BEGIN
   FOR j := eds to lds DO BEGIN
    summe := daten^.zahlen[i,j] + summe;
   END;
 END;
 startwinkel := 0;
 SetFillStyle(1,15);
 zaehler := 0;

 FOR i:= edb to ldb DO BEGIN
   FOR j := eds to lds DO BEGIN
     anteil := (daten^.zahlen[i,j]/summe)*360;
```

```
      endwinkel := startwinkel + Round(anteil);
      IF endwinkel > 360 THEN endwinkel := 360;
      mittelwinkel := Round((startwinkel+endwinkel)/2);
      xm := Round (xmitte+expl*cos(mittelwinkel*PI/180));
      ym := Round(ymitte - expl*verh*sin(mittelwinkel*PI/180));

      sector(xm,ym,Startwinkel,Endwinkel,xradius,Round(verh*xradius));

      halbierende := ((endwinkel+startwinkel)/2)*PI/180;
      IF eds = lds THEN BEGIN
        s := daten^.block[i];
      END
      ELSE BEGIN
        s := daten^.satz[j];
      END;
      beschrtorte(s,zaehler,anteil);
      inc(zaehler);
      startwinkel := endwinkel;
    END;
  startwinkel := endwinkel;
  END;
END;

{—— Tortendiagramm: Aufbau der 3-D-Tortengrafik          ——}
{—— Eingabe: erster Datensatz,letzter Datensatz,          ——}
{——          erster Datenblock, letzter Datenblock        ——}
{—— Ausgabe: Keine                                        ——}

procedure Tortendiagramm(edb,ldb : word; eds,lds : byte);
VAR xmitte : word;
anteil,summe,halbierende : single;
zaehler,startwinkel,endwinkel,untenwinkel,xm,ym : word;
i,j,k : word;
CONST
 ymitte : word = 23;       {Mittelpunkt y-Koordinate}
 xradius : word = 40;      {x-Radius EGA-Grafik}
 verh : single = 0.60;     {Verhältnis y-Radius zu x-Radius EGA}
 offset : word = 15;       {Dicke der Torte}
BEGIN
 xmitte := Grafinit+75;
 summe := 0;
 FOR i:= edb to ldb DO BEGIN
   FOR j := eds to lds DO BEGIN
    summe := daten^.zahlen[i,j] + summe;
   END;
 END;
 startwinkel := 0;
 zaehler := 0;
 ellipse(xmitte,ymitte,0,180,xradius,Round(xradius*verh));
 ellipse(xmitte,ymitte+offset,180,360,xradius,Round(xradius*verh));
 line(xmitte+xradius,ymitte,xmitte+xradius,ymitte+offset);
 line(xmitte-xradius,ymitte,xmitte-xradius,ymitte+offset);
 line(xmitte,ymitte,xmitte+xradius,ymitte);
 SetFillStyle(1,15);
```

```
zaehler := 0;

FOR i:= edb to ldb DO BEGIN
  FOR j := eds to lds DO BEGIN
    anteil := (daten^.zahlen[i,j]/summe)*360;
    endwinkel := startwinkel + Round(anteil);
    IF ((i <>ldb) or (j<>lds)) THEN BEGIN
      xm := xmitte+Round(xradius*cos(endwinkel*PI/180));
      ym := ymitte-Round(verh*xradius*sin(endwinkel*PI/180));
      line (xmitte,ymitte,xm,ym);
      IF (endwinkel >180) THEN line (xm,ym,xm,ym+offset);
    END;
    halbierende := ((endwinkel+startwinkel)/2)*PI/180;
    IF eds = lds THEN BEGIN
      s := daten^.block[i];
    END
    ELSE BEGIN
      s := daten^.satz[j];
    END;
    beschrtorte(s,zaehler,anteil);
    inc(zaehler);
    IF Round((i+j-eds-edb+1)/2) = (i+j-eds-edb+1)/2 THEN BEGIN
      FloodFill(xmitte+Round(xradius*cos(halbierende)/2),
                ymitte-Round(verh*xradius*sin(halbierende)/2),15);
      IF endwinkel > 180 THEN BEGIN
         SetColor(0);
         IF startwinkel < 180 THEN startwinkel := 180;
         ellipse(xmitte,ymitte,startwinkel,endwinkel,
                 xradius,Round(xradius*verh));
         SetColor(15);
      END;
    END
    ELSE BEGIN
      IF endwinkel > 180 THEN ellipse(xmitte,ymitte,
           startwinkel,endwinkel,xradius,Round(xradius*verh));
    END;
    startwinkel := endwinkel;
  END;
 startwinkel := endwinkel;
 END;
END;

{—— Balkenplot: Zeichnet die 2-D-Balken                    ——}
{—— Eingabe: Zeichenmasstab,Balkenhöhe,Startkoordinaten,  ——}
{——          Balkenbreite,aktueller und erster Datenblock ——}
{—— Ausgabe: Keine                                         ——}

procedure balkenplot(masstab :single; hoehe,xaktuell,yaktuell,
                     saeule,diff : word);
VAR k: word;
BEGIN
  bar(xaktuell,yaktuell-hoehe,xaktuell+saeule,yaktuell);
  IF hoehe > 0 THEN BEGIN
    CASE diff of
```

```
1 :  BEGIN
       SetColor(0);
       FOR k := xaktuell+1 to xaktuell+saeule-1 DO BEGIN
         line(k,yaktuell-hoehe+1,k,yaktuell-1);
       END;
       SetColor(15);
     END;
2 :    BEGIN
       SetColor(0);
       FOR k := xaktuell+1 to xaktuell+saeule-1 DO BEGIN
         line(k,yaktuell-hoehe+1,k,yaktuell-1);
       END;
       SetColor(15);
       IF saeule >=3 THEN BEGIN
         FOR k:= 1 to Trunc(saeule/3) DO BEGIN
           line (xaktuell+k*3,yaktuell,xaktuell+k*3,
                 yaktuell-hoehe+1);
         END;
       END;
     END;
3 :    BEGIN
       SetColor(0);
       FOR k := xaktuell+1 to xaktuell+saeule-1 DO BEGIN
         line(k,yaktuell-hoehe+1,k,yaktuell-1);
       END;
       SetColor(15);
       IF hoehe >= 6 THEN BEGIN
         FOR k:= 0 to Trunc(hoehe/6)-1 DO BEGIN
           line (xaktuell+1,yaktuell-6*k,
                 xaktuell+saeule-1,yaktuell-6*k-6);
         END;
       END;
     END;
4 :    BEGIN
       SetColor(0);
       FOR k := xaktuell+1 to xaktuell+saeule-1 DO BEGIN
         line(k,yaktuell-hoehe+1,k,yaktuell-1);
       END;
       SetColor(15);
       IF hoehe >=4 THEN BEGIN
         FOR k:= 1 to Trunc(hoehe/4) DO BEGIN
           line (xaktuell+1,yaktuell-4*k,xaktuell+
                 saeule-1,yaktuell-4*k);
         END;
       END;
     END;
5 :    BEGIN
       SetColor(0);
       FOR k := xaktuell+1 to xaktuell+saeule-1 DO BEGIN
         line(k,yaktuell-hoehe+1,k,yaktuell-1);
       END;
       SetColor(15);
       IF hoehe >= 6 THEN BEGIN
         FOR k:= 0 to Trunc(hoehe/6)-1 DO BEGIN
```

```
                    line (xaktuell+1,yaktuell-6*k-6,xaktuell+
                          saeule-1,yaktuell-6*k);
                  END;
                END;
              END;
      END;
   END;
END;

{--- Grafumgebung: Zeichnen und Beschriften Balkengrafik  ---}
{--- Eingabe: Begrenzungen links,unten,rechts,oben;Maximum ---}
{--- Ausgabe: Keine                                        ---}

procedure grafumgebung(links,unten,rechts,oben : word;
                       max : single);
VAR i : byte;
hoehe : word;
BEGIN
 Line(links,oben,links,unten);
 Line(rechts,oben,rechts,unten);
 FOR i:= 0 to 5 DO BEGIN
   hoehe := Round(i*((unten-oben)/5));
   Line(links-4,unten-hoehe,rechts,unten-hoehe);
   str(Round(max/5)*i,s);
   OutTextXY(links-6-length(s)*8,unten-hoehe,s);
 END;
END;

{--- Beschriftung: Beschriften der Balkengrafiken          ---}
{--- Eingabe: Begrenzungen links,unten,rechts,erster und   ---}
{---       letzter Datenblock,erster und letzter Datensatz ---}
{--- Ausgabe: Keine                                        ---}

procedure beschriftung(links,unten,rechts,edb,ldb,
                       eds,lds : word);
CONST
anz : array[1..6] of byte = (10,10,9,6,4,3);
VAR
i,xaktuell,yaktuell : word;
BEGIN
 xaktuell := links;
 yaktuell := unten+3;
 FOR i:= edb to ldb DO BEGIN
   CASE i-edb of
    0 :  bar(xaktuell,yaktuell,xaktuell+4,yaktuell+8);
    1..5 : BEGIN
            line(xaktuell,yaktuell,xaktuell+4,yaktuell);
            line(xaktuell,yaktuell,xaktuell,yaktuell+8);
            line(xaktuell,yaktuell+8,xaktuell+4,yaktuell+8);
            line(xaktuell+4,yaktuell+8,xaktuell+4,yaktuell);
            CASE i-edb of
             2:line(xaktuell+2,yaktuell,xaktuell+2,yaktuell+8);
             3:line(xaktuell,yaktuell+8,xaktuell+4,yaktuell);
             4:line(xaktuell,yaktuell+4,xaktuell+4,yaktuell+4);
```

```
                5:line(xaktuell,yaktuell,xaktuell+4,yaktuell+8);
               END;
              END;
    END;
    s := Copy(daten^.block[i],1,anz[(ldb-edb+1)]);
    OutTextXY(xaktuell+8,yaktuell,s);
    xaktuell := xaktuell + 11 + length(s)*6;
  END;
END;

{—— Balkendiagramm I : Aufbau der Balkengrafik seitlich ——}
{—— Eingabe: erster Datensatz,letzter Datensatz,erster    ——}
{——          Datenblock, letzter Datenblock               ——}
{—— Ausgabe: Keine                                         ——}

procedure Balkendiagramm1(edb,ldb : word; eds,lds : word);
VAR xmitte : word;
masstab,max : single;
hoehe, xstart, xaktuell, yaktuell, Saeule, Satzbreite,
 Blockbreite : word;
i,j,k : word;
CONST
 oben : word = 2;          {Obere Begrenzung}
 rechts : word = 239;      {Rechte Begrenzung EGA-Grafik}
 unten : word = 52;        {Untere Begrenzung}
 links : word = 45;        {Linke Begrenzung}
BEGIN
 xmitte := Grafinit;
 max := 0;
 FOR i:= edb to ldb DO BEGIN
   FOR j := eds to lds DO BEGIN
    IF daten^.zahlen[i,j] > max THEN max := daten^.zahlen[i,j];
   END;
 END;
 grafumgebung(links,unten,rechts,oben,max);
 masstab := (unten-oben)/max;
 Satzbreite := Trunc((rechts-links)/(lds-eds+1));
 Blockbreite := Trunc((Satzbreite*0.9)/(ldb-edb+1));
 Saeule := Round(Blockbreite*0.8);
 xstart := links+4;
 yaktuell := unten;
 FOR i:= eds to lds DO BEGIN
   xaktuell := xstart;
   FOR j := edb to ldb DO BEGIN
     hoehe := Round(daten^.zahlen[j,i]*masstab);
     balkenplot(masstab,hoehe,xaktuell,yaktuell,saeule,j-edb);
     xaktuell := xaktuell + blockbreite;
   END;
 xstart := xstart + satzbreite;
 END;
beschriftung(links,unten,rechts,edb,ldb,eds,lds);
END;

{—— Balkendiagramm II: Aufbau der gestapelten Balkengrafik ——}
```

```
{—— Eingabe: erster Datensatz,letzter Datensatz,erster      ——}
{——          Datenblock, letzter Datenblock                 ——}
{—— Ausgabe: Keine                                          ——}

procedure Balkendiagramm2(edb,ldb : word; eds,lds : word);
VAR xmitte : word;
masstab,max,summe : single;
hoehe, xstart, xaktuell, yaktuell, Saeule, Satzbreite : word;
i,j,k : word;
CONST
 oben : word = 2;       {Obere Begrenzung}
 rechts : word = 239;   {Rechte Begrenzung EGA-Grafik}
 unten : word = 52;     {Untere Begrenzung}
 links : word = 45;     {Linke Begrenzung}
BEGIN
 xmitte := Grafinit;
 max := 0;
 FOR j := eds to lds DO BEGIN
   summe := 0;
   FOR i:= edb to ldb DO BEGIN
     summe := summe + daten^.zahlen[i,j];
   END;
   IF summe > max THEN max := summe;
 END;
 masstab := (unten-oben)/max;
 grafumgebung(links,unten,rechts,oben,max);
 Satzbreite := Trunc((rechts-links)/(lds-eds+1));
 Saeule := Trunc(Satzbreite*0.80);
 xaktuell := links+4;
 FOR i:= eds to lds DO BEGIN
   yaktuell := unten;
   FOR j := edb to ldb DO BEGIN
     hoehe := Trunc(daten^.zahlen[j,i]*masstab);
     balkenplot(masstab,hoehe,xaktuell,yaktuell,saeule,j-edb);
     yaktuell := yaktuell -hoehe;
   END;
 xaktuell := xaktuell + satzbreite;
 END;
beschriftung(links,unten,rechts,edb,ldb,eds,lds);
END;

{—— Saeulendiagramm: Aufbau der 3-D-Saeulengrafik           ——}
{—— Eingabe: erster Datensatz,letzter Datensatz,erster      ——}
{——          Datenblock, letzter Datenblock                 ——}
{—— Ausgabe: Keine                                          ——}

procedure Saeulendiagramm(edb,ldb : word; eds,lds : byte);
VAR xmitte,zahl,xoff,unt : word;
s : string;
masstab,max : single;
hoehe, xstart, xaktuell, yaktuell, Saeule,
 Satzbreite : Integer;
i,j,k : word;
CONST
```

```
 oben : word = 2;           {Obere Begrenzung}
 rechts : word = 200;       {Rechte Begrenzung EGA-Grafik}
 unten : word = 50;         {Untere Begrenzung}
 hinten : word = 20;        {Hintere Begrenzung}
 diff : word = 10;          {y-Offset für Datenblock}
 verh   : single = 0.7;     {Verhältnis Tiefe Säulen - y-Offset}

BEGIN
 xmitte := Grafinit;
 max := 0;
 FOR i:= edb to ldb DO BEGIN
   FOR j := eds to lds DO BEGIN
    IF daten^.zahlen[i,j] > max THEN max := daten^.zahlen[i,j];
   END;
 END;
 masstab := (unten-oben)/max;
 Line(hinten,unten,rechts,unten);
 Line(hinten,oben,rechts,oben);
 Line(hinten,unten,hinten,oben);
 Line(rechts,unten,rechts,oben);
 Line(hinten,oben,hinten-diff,oben+diff);
 Line(hinten-diff,oben+diff,hinten-diff,
      unten+diff);
 Line(hinten-diff,unten+diff,hinten,unten);
 Line(hinten-diff,unten+diff,rechts-diff,
      unten+diff);
 Line(rechts-diff,unten+diff,rechts,unten);
 FOR i:= 0 to 5 DO BEGIN
   hoehe := Trunc(i*((unten-oben)/5));
   Line(rechts+4,unten-hoehe,hinten,unten-hoehe);
   Line(hinten,unten-hoehe,hinten-diff,
        unten+diff-hoehe);
   str(Round(max/5)*i,s);
   IF i<> 0 THEN OutTextXY(rechts+6,unten-hoehe,s);
 END;

 IF eds = lds THEN BEGIN
   unt := (ldb-edb+1);
 END
 ELSE BEGIN
   unt := (lds-eds+1);
 END;
 Satzbreite := Trunc((rechts-hinten-10)/unt);
 Saeule := Trunc(Satzbreite*0.7);
 xstart := hinten+10;
 xoff := Trunc(Verh*diff);

 xaktuell := xstart-diff;
 unt := unten+diff;
 FOR i := edb to ldb DO BEGIN
   FOR j := eds to lds DO BEGIN
     hoehe := Trunc(daten^.zahlen[i,j]*masstab);
     bar(xaktuell,unt-hoehe,xaktuell+saeule,unt);
     SetColor(0);
```

```
      FOR k := 1 to xoff-1 DO BEGIN
       line (xaktuell+saeule+k,unt-k,xaktuell+saeule+k,
             unt-hoehe-k);
      END;
      FOR k := 1 to saeule DO BEGIN
       line (xaktuell+k,unt-hoehe,xaktuell+k+xoff,
             unt-hoehe-xoff);
      END;
      SetColor(15);
      line(xaktuell+saeule,unt,xaktuell+saeule+xoff,unt-xoff);
      line(xaktuell+saeule,unt-hoehe,xaktuell+saeule+xoff,
           unt-hoehe-xoff);
      line(xaktuell,unt-hoehe,xaktuell+xoff,unt-hoehe-xoff);
      line(xaktuell+saeule+xoff,unt-xoff,xaktuell+saeule+xoff,
           unt-hoehe-xoff);
      line(xaktuell+xoff,unt-hoehe-xoff,xaktuell+saeule+xoff,
           unt-hoehe-xoff);

      xaktuell := xaktuell + satzbreite;
    END;
  END;
  xaktuell := rechts+4;
  yaktuell := unten+5;
  IF eds = lds THEN BEGIN
     s := daten^.block[eds];
  END
  ELSE BEGIN
     s := daten^.satz[edb];
  END;
  OutTextXY(xaktuell,yaktuell,Copy(s,1,5));
END;

{—— Mini: Berechnet Minimum zweier Zahlen                  ——}
{—— Eingabe: Zu vergleichende Zahlen                       ——}
{—— Ausgabe: Minimum dieser Zahlen                         ——}

function mini(erste,zweite : word) : word;
BEGIN
 IF erste <= zweite THEN BEGIN
   mini := erste;
 END
 ELSE BEGIN
   mini := zweite;
 END;
END;

{—— Auswahlblsa: Auswahl der darzustellenden               ——}
{——                  Datenblöcke/Sätze                     ——}
{—— Eingabe: Maximal zulässige Anzahl Datenblöcke/sätze,   ——}
{——          Typ (Blöcke/Sätze) mit entsp. Textausgaben    ——}
{—— Ausgabe: Setzt globale Variablen eblo,anzbl/esa,anzsa  ——}

procedure auswahlblsa(max : byte; art,leer : string;
                      loadart : word; VAR erster,anzahl : word);
```

```
VAR code : integer;
s1 : string;
BEGIN
  Standardbild;
  GotoXY(2,2);
  write('Erster Daten',art,' (1-',loadart,') : <',leer,'>');
  str(loadart,s1);
  REPEAT
    Eingabe(s,23+length(art)+length(s1),2,length(leer),1,'1');
    val(s,erster,code);
  UNTIL ((erster >= 1) and (erster <= loadart));
  IF max > 1 THEN BEGIN
    IF erster < loadart THEN BEGIN
      GotoXY(2,4);
      str(erster,s1);
      str(Mini(erster+max-1,loadart),s);
      s1 := s1 + '-' + s;
      write('Letzter Daten',art,' (',s1,') : <',leer,'>');
      REPEAT
        Eingabe(s,22+length(art)+length(s1),4,length(leer),1,'');
        val(s,anzahl,code);
      UNTIL ((anzahl >=erster) and
            (anzahl <=Mini(erster+max-1,loadart)));
    END
    ELSE BEGIN
      anzahl := loadart;
    END;
  END
  ELSE BEGIN
    anzahl := erster;
  END;
  anzahl := anzahl - erster+1;
END;

{-- Auswahlblock: Auswahl der darzustellenden Datenblöcke --}
{-- Eingabe: Maximal zulässige Anzahl Datenblöcke         --}
{-- Ausgabe: Setzt globale Variablen eblo,anzbl           --}

procedure auswahlblock(max : byte);
BEGIN
 auswahlblsa(max,'block','   ',loadblock,eblo,anzbl);
END;

{-- Auswahlsatz: Auswahl der darzustellenden Datensätze  --}
{-- Eingabe: Maximal zulässige Anzahl Datensätze         --}
{-- Ausgabe: Setzt globale Variablen esa,anzsa           --}

procedure auswahlsatz(max : byte);
BEGIN
 auswahlblsa(max,'satz','   ',loadsatz,esa,anzsa);
END;

{-- ausdruck: Hardcopy der Grafik auf Matrixdrucker   --}
{-- Eingabe: Keine                                    --}
```

```
{—— Ausgabe: Keine                                          ——}

procedure ausdruck;
CONST
 ret : char = chr(13);
 druckinit = #$0D#$1B#$6C#$08#$1B#$41#$07;
 neuezeile = #$0D#$0A;
 grafikzeile = #$1B#$4C#$80#$00;
VAR
 f : file of char;
 grafikbyte : array[0..63] of byte;
 i,j : integer;
BEGIN
  Assign(f,'prn');
  Rewrite(f);
  write(f,ret);
  IF IOResult = 0 THEN BEGIN
     close(f);
     write(lst,druckinit);
     FOR i:= 0 to 29 DO BEGIN
      write(lst,grafikzeile);
      FOR j:= 0 to 63 DO BEGIN
        grafikbyte[j] := Mem[$B000:1890-j*30+i];
      END;
      FOR j:= 0 to 63 DO BEGIN
        write(lst,chr(grafikbyte[j]));
        write(lst,chr(grafikbyte[j]));
      END;
      write(lst,neuezeile);
    END;
    write(lst,neuezeile);
  END;
END;

{————————————————————————————————————————————————————————————}
{——————————————   P R O G R A M M S T A R T   ——————————————}
{————————————————————————————————————————————————————————————}

BEGIN
   New(daten);
   letztpfad := '';
   letztname := '';
   loadsatz := 0;              {Anzahl geladene Datensätze}
   loadblock := 0;             {Anzahl geladene Datenblöcke}
   druck := 1;                 {Keine Hardcopy nach Bildaufbau}

{——————————————— Hauptmenü aufbauen ————————————————————————}

  Textmode(portfolio);
  Standardbild;
  REPEAT
   GotoXY(2,2);
   write('ATARI-Portfolio-Version 1.1');
   GotoXY(2,3);
```

```
write('Juli 1990 von Frank Riemenschneider');
GotoXY(2,4);
write('(C) 1990 Markt & Technik Verlag AG');
GotoXY(2,6);
write('Hauptmenü mit <F1>  Ausdruck : ',hardcopy[druck]);
REPEAT
  input := ReadKey;
  IF input = chr(0) THEN input := ReadKey;
UNTIL Input = chr(59);
hpunkt := portmenue(' Hauptmenü ',12,2,4,16,hauptmenue);
CASE hpunkt of
  1 : BEGIN
        punkt := portmenue(' Datenmenü ',10,2,3,20,datenmenue);
        CASE punkt of
             1 : einrichten;
             3 : einladen(false);
             2 : einladen(true);
        END;
        IF punkt <>0 THEN Standardbild;
      END;
  2 : BEGIN
        IF ((loadblock >0) and (loadsatz > 0)) THEN BEGIN
          punkt := portmenue(' Grafikmenü ',10,1,5,19,grafikmenue);
          CASE punkt of
               1 : BEGIN
                     auswahlblock(6);
                     auswahlsatz(Trunc(20/anzbl));
                     balkendiagramm1(eblo,eblo+anzbl-1,
                                     esa,esa+anzsa-1);
                   END;
               3 : BEGIN
                     auswahlblock(20);
                     IF anzbl > 1 THEN BEGIN
                       auswahlsatz(1);
                     END
                     ELSE BEGIN
                       auswahlsatz(20);
                     END;
                     saeulendiagramm(eblo,eblo+anzbl-1,
                                   esa,esa+anzsa-1);
                   END;
               2 : BEGIN
                     auswahlblock(6);
                     auswahlsatz(20);
                     balkendiagramm2(eblo,eblo+anzbl-1,
                                    esa,esa+anzsa-1);
                   END;
               4 : BEGIN
                     auswahlblock(8);
                     IF anzbl > 1 THEN BEGIN
                        auswahlsatz(1);
                     END
                     ELSE BEGIN
                        auswahlsatz(8);
```

```
                END;
                kuchendiagramm(eblo,eblo+anzbl-1,
                               esa,esa+anzsa-1);
              END;
          5 : BEGIN
                auswahlblock(8);
                IF anzbl > 1 THEN BEGIN
                  auswahlsatz(1);
                END
                ELSE BEGIN
                  auswahlsatz(8);
                END;
                tortendiagramm(eblo,eblo+anzbl-1,
                               esa,esa+anzsa-1);
              END;
        END;
        IF ((punkt >0) and (punkt<6)) THEN BEGIN
          IF druck = 0 THEN ausdruck;
          REPEAT;
          UNTIL KeyPressed;
          s[1] := ReadKey;
          CloseGraph;
          Standardbild;
        END;
      END;
    END;
  3 : druck := 1-druck;
  END;
 UNTIL hpunkt = 4;
 ClrScr;
 Dispose(daten);
END.
```

2.7 Ein Backup-Programm für RAM-Karten

Als besonders bitter empfinde ich das Fehlen des DOS-Befehls DISKCOPY für die RAM-Karten. Wenn ich mir eine Sicherheitskopie anlegen möchte, muß ich immer mit dem umständlichen und langsamen COPY arbeiten, davon abgesehen, daß eventuell versteckte Daten gar nicht erst mitkopiert werden. Versteckte Daten?

Nun, z.B. die Realisation eines Kopierschutzes auf RAM-Karten ist nicht möglich. Die einzige Möglichkeit, Daten vor dem Vervielfältigen zu schützen, besteht gerade darin, daß durch den COPy-Befehl Daten, die sich in Sektoren befinden, die nicht durch eine Datei belegt sind, auch nicht mitkopiert werden. Sicherlich kann man sagen, daß es z.Z. noch

wenig Software für den Portfolio gibt, schon gar keine kopiergeschützte, aber irgendwann kann ja auch einmal ein Softwarehaus auf diese Idee kommen. Der zweite Punkt sind die versteckten Daten. Viele Programme benötigen Daten, um ständig auf dem aktuellen Stand zu sein. Dafür legen Sie entweder eine für jedermann sichtbare Datei an, oder Sie verstecken diese Daten, wenn sie z.B. zuviel über die interne Arbeitsweise des Programms verraten, in einem vorher allokierten Sektor, der nicht zur Datei gehört. Bei einer Kopie mit COPY blieben diese Daten unkopiert, was sicherlich nicht in Ihrem Sinn wäre.

Als weiteres Problem muß man mit COPY den Umweg über die interne RAM-Disk gehen. Wenn diese nun zu klein geraten ist, d.h. weniger Speicher aufweist als die Dateien der RAM-Karte, muß man in mehreren Durchgängen kopieren, was sehr unangenehm sein kann, da der Joker (x) nicht mehr universell einsetzbar ist. (Sonst kopiert man ja jedesmal wieder alles...)

Um all diesen Fragen ein Ende zu bereiten, habe ich das Backup-Programm RAM-COPY entwickelt, das sämtliche Sektoren einer RAM-Karte kopiert, unabhängig davon, ob ein Sektor durch eine Datei belegt ist oder nicht. Das Programm kopiert 32- und 64-Kbyte-Disks in einem Durchgang, die 128-Kbyte-Disk in zwei Durchgängen. Der Kopiervorgang wird dabei ungleich schneller durchgeführt als bei dem Einsatz des COPy-Befehls, da direkt auf die einzelnen Sektoren zugegriffen wird. Als zusätzlichen Komfort können Sie Start- und Zielspur frei einstellen (Aufbau einer RAM-Karte siehe Kapitel 1), so daß auch Ausschnitte der Disk kopiert werden können.

Nach dem Programmstart mit

BACKUP [Return]

erscheint ein Menü mit folgenden Wahlmöglichkeiten:

[C]: Startet den Kopiervorgang
[F]: Legt fest, ob Zieldisk beim Kopieren formatiert werden soll
[E]: Programmende
[←]: Verringert Startspur (0 bis 31 einstellbar)
[→]: Erhöht Startspur (0 bis 31 einstellbar)
[↓]: Erniedrigt Endspur (0 bis 31 einstellbar)
[↑]: Erhöht Endspur (0 bis 31 einstellbar)

Dabei ist zu beachten, daß die Endspur nicht größer sein darf als die Startspur. Wenn der Kopiervorgang gestartet wird, erscheint die Meldung

Quellkarte einlegen, die Sie mit [Return] bestätigen müssen. Nun werden die Daten eingelesen. Anschließend wird die Meldung *Zielkarte einlegen* ausgegeben, die Sie nach dem Wechsel der Karte ebenfalls mit [Return] bestätigen müssen. Bei der 128-Kbyte-Karte wiederholt sich diese Prozedur noch einmal. Nach Beendigung des Kopiervorgangs sollte dann wieder das Hauptmenü erscheinen. Je nach Lage der Dinge kann aber auch eine Fehlermeldung erscheinen, wobei insgesamt acht Variationen möglich sind:

»Falsche Funktionsnummer«, »Keine Adreßmarkierung«, »Karte schreibgeschützt«, »Sektor nicht gefunden«, »DMA – Überlauf«, »Segmentgrenze-Überlauf«, »Lesefehler«, »Fehler des Controllers«, »Track nicht gefunden«,»Laufwerk reagiert nicht«.

Die Bedeutung und Behebung der Fehlermeldungen können Sie in jedem DOS-Handbuch nachlesen.

```
PROGRAM Backup;
{written by Frank Riemenschneider
            Postfach 730309
            3000 Hannover 71}

USES DOS,PORTCRT;

{$M 2000,65536,65536}

TYPE
buf = array[1..65535] of byte;     {Puffer für Magnetkarten-Daten}

VAR
groesse,segment,i : word;
puffer : ^buf;
s : char;
erster,letzter,status,starttrack,endtrack,ascii : byte;
form : string[4];

{— Bearbtrack: Liest oder schreibt Sektor der Magnetkarte ———}
{— Eingabe: Modus (2=lesen, 3=schreiben), Track, Sektor   ———}
{— Ausgabe: Error-Code (0=Fehlerfrei)                     ———}

FUNCTION bearbtrack(modus,track,sektor:byte) : byte;
VAR cylinder,seite : byte;
reg : registers;
position : word;
BEGIN
  seite := 0;
  seite := Trunc(track/16);          {Karten'Seite' (0 oder 1)}
  cylinder := track - 16*seite;      {Karten'Zylinder' (0 bis 15)}
  reg.ah := modus;
```

```
  reg.dl := 0;
  reg.dh := seite;
  reg.ch := cylinder;
  reg.cl := sektor;
  reg.al := 1;                    {1 Sektor lesen}
  reg.es := Segment;
  position := (track-erster)*groesse*8+(sektor-1)*groesse;
  reg.bx := position;             {Pufferpositon des Sektors berechnen}
  Intr($13,reg);
  bearbtrack := reg.ah;           {Error-Code zurückgeben}
END;

{— Maketrack: Liest/Beschreibt/Verifiziert einen Track       ——}
{— Eingabe: Tracknummer, Modus (2=L,3=S,4=V)                 ——}
{— Ausgabe: Error-Code (0=Fehlerfrei)                        ——}

FUNCTION maketrack(track,modus : byte) : byte;
VAR status,i : byte;
BEGIN
  i := 1;
  status := 0;
  WHILE ((i<= 8) and (status=0)) do BEGIN
    status := bearbtrack(modus,track,i);
    inc(i);
  END;
  maketrack := status;
END;

{— Formattrack: Formatiert einen Track                       ——}
{— Eingabe: Tracknummer                                      ——}
{— Ausgabe: Error-Code (0=Fehlerfrei)                        ——}

FUNCTION Formattrack(track : byte) : byte;
VAR modus : byte;
BEGIN
   modus := 5;
   IF track = 0 THEN modus := $83;
   Formattrack := bearbtrack(modus,track,1);
END;

{— Leseein: Liest einen Bereich der Magnetkarte              ——}
{— Eingabe: Erster Track, letzter Track                      ——}
{— Ausgabe: Error-Code (0=Fehlerfrei)                        ——}

FUNCTION leseein(erster,letzter : byte) : byte;
VAR i,status : byte;
BEGIN
  status := 0;
  i := erster;
  WHILE ((status = 0) and (i<=letzter)) do BEGIN
     status := maketrack(i,2);
     inc(i);
  END;
  leseein := status;
```

```
END;

{— Schreibeauf: Beschreibt einen Bereich der Magnetkarte  ——}
{— Eingabe: Erster Track, letzter Track                   ——}
{— Ausgabe: Error-Code (0=Fehlerfrei)                     ——}

FUNCTION schreibeauf(erster,letzter : byte) : byte;
VAR i,status : byte;
reg : registers;
BEGIN
  status := 0;
  IF (form = '  Ja') THEN BEGIN
    i := erster;
    WHILE ((status = 0) and (i<=letzter)) do BEGIN
       status := Formattrack(i);
       inc(i);
    END;
  END;
  i := erster;
  WHILE ((status = 0) and (i<=letzter)) do BEGIN
     status := maketrack(i,3);
     inc(i);
  END;
  WHILE ((status = 0) and (i<=letzter)) do BEGIN
     status := maketrack(i,4);
     inc(i);
  END;
  schreibeauf := status;
END;

{———————————— Start Hauptprogramm ————————————}

BEGIN
 New(puffer);
 groesse := 128;
 endtrack := 31;
 starttrack := 0;
 form := 'Nein';
 Segment := Memw[seg(puffer):ofs(puffer)+2];
 ClrScr;
 GotoXY(0,0);
 writeln('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx');
 writeln('x     C C M  -  C O P Y  V  1 . 0    x');
 writeln('x (C) 1990 Markt & Technik Verlag AG  x');
 writeln('x   written by Frank Riemenschneider  x');
 writeln('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx');
 writeln('x Starttrack : <  >  Endtrack : <  >  x');
 writeln;
 write('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx');
 REPEAT
   GotoXY(0,6);
   write('x <C>opy <F>ormat Ziel:       <E>nde  x');
   GotoXY(16,5);
   write('  ');
```

```
GotoXY(16,5);
write(starttrack);
GotoXY(33,5);
write('  ');
GotoXY(33,5);
write(endtrack);
GotoXY(24,6);
write(form);
REPEAT
UNTIL Keypressed;
s := UpCASE(readKey);
ascii := Ord(s);
CASE ascii of
  0 : BEGIN
        s := ReadKey;
        CASE s of
          chr(80) : IF starttrack > 0 THEN dec(starttrack);
          chr(72) : IF ((starttrack < 31) and (starttrack < endtrack)) THEN
                                                                 inc(starttrack);
          chr(75) : IF ((endtrack > 0) and (endtrack > starttrack)) THEN dec(endtrack);
          chr(77) : IF endtrack < 31 THEN inc(endtrack);
        END;
       END;
  67 : BEGIN
         GotoXY(2,6);
         write('Bitte Q U E L L - Karte einlegen   ');
         s := ReadKey;
         erster := 0;        {Boot-Sektor lesen}
         letzter := 0;
         status := leseein(erster,letzter);
         IF status = 0 THEN BEGIN
           groesse := Memw[Segment:11];  {Kartengröße holen}
           erster := starttrack;
           letzter := endtrack;
           REPEAT
             IF ((groesse = 512) and (letzter > erster+15)) THEN
                letzter := erster + 15;
             IF erster > starttrack THEN BEGIN
               GotoXY(2,6);
               write('Bitte Q U E L L - Karte einlegen   ');
               s := ReadKey;
             END;
             GotoXY(2,6);
             write('Lese Daten ein .......          ');
             status := leseein(erster,letzter);
             IF status = 0 THEN BEGIN
               GotoXY(2,6);
               write('Bitte Z I E L - Karte einlegen      ');
               s := ReadKey;
               GotoXY(2,6);
               write('Schreibe Daten .....               ');
               status := schreibeauf(erster,letzter);
             END;
             erster := letzter+1;
```

```
            letzter := endtrack;
          UNTIL ((status <>0) or (erster > endtrack));
        END;
        GotoXY(2,6);
        IF status = 0 THEN BEGIN
          write('Kopiervorgang abgeschlossen!      ');
        END
        ELSE BEGIN
          write('Fehler : ');
          CASE status of
            $01 : write('Falsche Funktionsnummer');
            $02 : write('Keine Adreßmarkierung  ');
            $03 : write('Karte schreibgeschützt ');
            $04 : write('Sektor nicht gefunden  ');
            $08 : write('DMA - Überlauf         ');
            $09 : write('Segmentgrenze-Überlauf ');
            $10 : write('Lesefehler             ');
            $20 : write('Fehler des Controllers ');
            $40 : write('Track nicht gefunden   ');
            $80 : write('Laufwerk reagiert nicht');
          END;
        END;
        s := ReadKey;
      END;
      70 : BEGIN
             IF form = 'Nein' THEN BEGIN
               form := '  Ja';
             END
             ELSE BEGIN
               form := 'Nein';
             END;
           END;
    END;
  UNTIL ascii = 69;
  ClrScr;
  Dispose(puffer);
END.
```

2.8 TSR-Programmierung auf dem Portfolio

Eine der wohl faszinierendsten Programmiermethoden eines PC stellt die sogenannte TSR-Programmierung dar. TSR steht für *Terminate and Stay Resident*, d.h., die entsprechenden Programme verbleiben nach ihrer Beendigung im Speicher und arbeiten dort entweder im Hintergrund weiter oder werden per Tastendruck aktiviert (Hot-Key). Der bekannteste Vertreter dieser Gattung von Programmen ist wohl das Programm

SIDEKICK. Nun, wenn Sie denken, irgendein auf dem PC entwickeltes TSR-Programm liefe auf dem Portfolio, so muß ich Ihnen leider sagen, daß dies ein gewaltiger Irrtum ist, zumindest, wenn das Programm über Hot-Key aktiviert werden soll. Warum dies so ist, werde ich Ihnen in Abschnitt 2.8.3 verraten, zuerst aber möchte ich Ihnen an Hand eines einfachen Beispiels die Grundlagen der TSR-Programmierung erläutern.

2.8.1 Der DOS-EXEC-Loader

Bevor wir in die Beispielprogramme einsteigen, möchte ich ein paar Worte über den Startvorgang eines DOS-Programms verlieren. Sämtliche DOS-Programme werden mit Hilfe der EXEC-Funktion geladen und gestartet. Sie wird nicht nur vom Kommandoprozessor benutzt, um externe Befehle oder Programme aufzurufen, sondern kann auch vom Anwender innerhalb eines Programms aufgerufen werden. Zunächst wird über andere DOS-Funktionen Speicherplatz im RAM reserviert, in den das Programm eingeladen werden kann. Am Anfang dieses Speichers legt DOS eine Struktur mit dem Namen PSP (Program Segment Prefix) an, an deren Ende dann das Programm geladen wird. Nach der Initialisierung des Stack und der Segmentregister wird das Programm schließlich zur Ausführung gebracht. Nach Ende der Programmausführung wird der durch den PSP und den Programmcode belegte Speicher normalerweise wieder freigegeben, wenn hierzu die DOS-Funktion $4C benutzt wird. Wenn man hingegen die Funktion $31 verwendet, wird das Programm zwar ebenfalls beeendet, der Speicher bleibt jedoch nach wie vor reserviert, so daß er durch kein anderes Programm belegt werden kann. In diesem Fall spricht man von einem TSR-Programm.

Eine wichtige Rolle spielt das PSP, so daß wir uns noch etwas ausführlicher mit ihm beschäftigen müssen. Die Struktur ist 256 Byte lang (16 Paragraphen) und enthält folgende Einträge:

Aufbau der Program-Segment-Prefix-Struktur

Adresse	Inhalt	Größe
$00	Befehl Int $20	2 Byte
$02	Segmentadresse Speicherende	2 Byte
$04	Nicht belegt	1 Byte
$05	FAR-CALL-Befehl zum Int $21	5 Byte

Aufbau der Program-Segment-Prefix-Struktur

Adresse	Inhalt	Größe
$0A	Kopie des Interrupt-Vektors $22	4 Byte
$0E	Kopie des Interrupt-Vektors $23	4 Byte
$12	Kopie des Interrupt-Vektors $24	4 Byte
$16	Nicht belegt	22 Byte
$2C	Segmentadresse Environment-Block	2 Byte
$2E	Nicht belegt	46 Byte
$5C	File-Control-Block 1	16 Byte
$6C	File-Control-Block 2	16 Byte
$80	Anzahl Zeichen in Kommandozeile	1 Byte
$81	Kommandozeile	127 Byte

Das PSP entstammt den ersten DOS-Versionen, so daß viele Einträge heute keinerlei praktische Bedeutung mehr haben. Der erste Eintrag ist ein schlichter Int-$20-Befehl, mit dem ein Programm beendet werden kann. Heute nimmt man im allgemeinen die DOS-Funktion $4C des Interrupt $21. Wichtig hingegen ist der folgende Eintrag: In ihm ist das Segment der Endadresse des für das Programm reservierten Speichers abgelegt. Hiermit kann ein Programm z.B. feststellen, ob es noch zusätzlichen Speicher z.B. für einen Heap reservieren kann, ohne daß dieser über eine DOS-Funktion allokiert werden müßte. Der nächste Eintrag ist wieder antiquiert. Es handelt sich um einen Aufruf des Interrupt $21, nicht jedoch mit Hilfe des INT-Befehls, sondern über einen direkten Einsprung mittels eines FAR-CALL-Befehls. Da hier jedoch nur Funktionen bis einschließlich $24 aufgerufen werden können und zudem die Registerbelegung von der des INT-Aufrufs abweicht, macht von dieser Möglichkeit niemand mehr Gebrauch.

Die folgenden drei Einträge enthalten Kopien der Interrupt-Vektoren für die Programmbeendigung ($22), für die Reaktion auf die Tastenkombinationen Control-C oder Control-Break ($23) und für die DOS-Fehler-Routine ($24). Beim Programmstart werden die aktuellen Werte in das PSP kopiert. Wenn diese Vektoren nun von einem Programm verändert werden, kann DOS nach dem Programmende an Hand der PSP-Einträge die ursprünglichen Einträge wiederherstellen.

Die Segmentadresse des sogenannten Environment-Blocks ist im folgenden Eintrag vermerkt, die Offsetadresse des Environment-Blocks ist relativ zu dieser Segmentadresse immer Null. Beim Environment-Block handelt

es sich einfach um eine Folge von ASCII-Strings, die jeweils mit einem Null-Zeichen abgeschlossen sind. Einer dieser Strings enthält z.B. das Verzeichnis, in dem sich die Datei COMMAND.COM befindet, andere die Suchpfade, die mit dem PATH-Befehl festgelegt wurden.

Es folgen nun die beiden sogenannten File-Control-Blöcke (FCB). Hierzu muß man wissen, daß DOS zwei Arten von Zugriffen auf Dateien gestattet: Zum einen über die sogenannten *Handle*-Funktionen (Funktionen $3C bis $5A des Interrupt $21), die heute fast ausschließlich verwendet werden (beim Öffnen der Datei wird ihr eine Nummer, *Handle*, zugeordnet, über die alle weiteren Zugriffe erfolgen), sowie aus Kompatibilitätsgründen zu dem Uralt-Betriebssystem CP/M über die FCB (Funktionen $0F bis $29 des Interrupt $21). Dabei wird in einen der beiden FCBs der Dateiname eingetragen, der anhand der Segment- und Offsetadresse den DOS-Funktionen übergeben wird. Leider unterstützt diese Methode kein hierarchisches Dateisystem, wie man es heute gewohnt ist, so daß die beiden FCBs heute praktisch bedeutungslos sind. Die ersten beiden Parameter beim Programmaufruf werden automatisch in die beiden FCBs eingetragen, was heute aber auch eher uninteressant ist.

In den 127 Byte großen Puffer für die Kommandozeile hingegen werden alle Parameter eingetragen, also auch die, die schon in den beiden FCBs gespeichert wurden. Eine Ausnahme bilden die Parameter zur Umleitung der Ein-/und/oder Ausgabe, da diese nichts mit dem Programm zu tun haben, sondern vom Betriebssystem direkt verarbeitet werden müssen. Da man nicht immer davon ausgehen kann, daß der Puffer komplett gefüllt ist, wird die Anzahl der Zeichen inklusiv des abschließenden Carriage-Return in dem vorletzten Eintrag des PSP festgehalten.

Bei Dateizugriffen über FCBs wird der Parameterpuffer als Ein-/Ausgabepuffer für Daten verwendet, den man dann als DTA (Disk Transfer Area) bezeichnet. Das Programm kann allerdings die DAT mit Hilfe der DOS-Funktion $1A in einen anderen Bereich verschieben.

Was hat nun das PSP mit TSR-Programmen zu tun? Nun, sehr viel: Wenn ein Programm resident im Speicher verbleibt, bleibt dort neben dem Programmcode natürlich auch das PSP, der Environment-Block und die DTA. Falls man nun das Programm irgendwann wieder aus dem Speicher entfernen möchte, genügt es nicht, den von DOS reservierten Speicher freizugeben, mit dem dann das PSP und der Programmcode eliminiert werden, sondern man muß zusätzlich den Environment-Speicher und im Zweifelsfall den DTA-Speicher freigeben, da das Programm diesen ja theo-

retisch aus dem PSP in irgendeinen anderen Bereich verschoben haben könnte, der für diesen Zweck allokiert wurde.

Man sollte es sich daher angewöhnen, bei wirklich jedem TSR-Programm diese drei Komponenten (PSP + Code, Environment, DTA) explizit freizugeben, auch wenn man genau weiß, daß z.B. keine DAT außerhalb des PSP vorhanden ist. Es kann nämlich vorkommen, daß man irgendwann das Programm so ändert, daß die DTA doch verschoben wird, ohne daß man sich daran erinnert, daß dieser Bereich dann auch freigegeben werden muß. Und schon hat man Speichermüll, der wertvolle Bytes stiehlt.

2.8.2 Verhinderung des Starts der Applikationen

Als erstes Beispielprogramm habe ich eine kleine Routine entwickelt, die den Start der Applikationen mit Hilfe der [Atari]-Taste (das ist die Taste mit dem Atari-Logo) verhindert. Ich habe mich nämlich schon oft darüber geärgert, daß ein von mir entwickeltes Programm nach dem Aufruf einer Applikation abstürzt, da diese zumindest in der DIP-OS-Version 1.052 noch diverse Fehler enthalten. Nach dem Aufruf des Programms NOATARI5.EXE bzw. von NOATARI7.EXE bleibt dieses solange resident im Speicher, bis es durch einen erneuten Aufruf wieder entfernt wird.

Eine wichtige Rolle spielt der Hardware-Tastatur-Interrupt $09. Dessen Service-Routine prüft nämlich, ob die [Atari]-Taste betätigt wurde und ruft die ausgewählte Applikation auf. Der Vektor wird einfach auf eine eigene Routine umgelegt, die nun ihrerseits prüft, ob die [Atari]-Taste betätigt wurde. Ist dies der Fall, wird der Tastaturcode durch einen anderen, »harmlosen« ersetzt, bevor wieder in die Original-Routine gesprungen wird. Dies mußte mittels *DB*-Anweisungen geschehen, da der Turbo Assembler keine absoluten Sprünge mit Segment- und Offsetadresse zuläßt.

Beim ersten Aufruf des Programms wird zunächst ein Flag gesetzt, das anzeigt, daß das Programm installiert wurde. Dann werden der alte Interruptvektor $09 und die Segmentadresse des PSP gerettet. Letztere wird benötigt, um bei der Wiederentfernung an die Segmentadresse des Environment-Blocks zu kommen. Schließlich wird der neue Vektor gesetzt und das Programm durch die Funktion $31 resident beendet.

Beim zweiten Aufruf wird zunächst das Flag überprüft, ob das Programm bereits installiert ist. Dazu bedient man sich eines Tricks: Da das Programm beim zweiten Aufruf natürlich auch ein zweitesmal durch den

EXEC-Loader in den Speicher geladen wird (es wird natürlich *nicht* das sich bereits im Speicher befindliche residente Programm zum zweitenmal gestartet!), kann man nicht einfach auf die Variable *flag* zugreifen, da diese natürlich auch zweimal vorhanden ist. Wie bekommt man nun das Codesegment des ersten, resident installierten Programms heraus? Ganz einfach: über den Interrupt-Vektor. Er zeigt natürlich in den Programmcode des residenten Programms. Jetzt verstehen Sie auch, warum die Variablen über das Code- und nicht über das Datensegment adressiert wurden! Nur das Codesegment kann man auf so einfache Weise herausfinden.

Ist das Programm schon installiert, werden der Programmspeicher sowie der Environment-Speicher des residenten Programms freigegeben. Anschließend wird das Programm über die Funktion $4C beendet, wodurch auch alle Speicherbereiche des beim zweitenmal aufgerufenen Programms freigegeben werden.

Dieses Verfahren zur Wiederentfernung von TSR-Programmen durch einen zweiten Aufruf des gleichen Programms hat sich inzwischen auch im professionellen Bereich immer mehr durchgesetzt.

Da die BIOS-Versionen 1.052 und 1.072 einen unterschiedlichen Interrupt-Handler $09 aufweisen, mußte das Programm in zwei Versionen erstellt werden. Hier zunächst das Listing für die Version 1.052:

```
; Atari-DEAKTIVATOR BIOS 1.052
; written by Frank Riemenschneider
; Postfach 730309
; 3000 Hannover 71
; Beim ersten Aufruf wird die Atari-Taste deaktiviert,
; so daß keine Applikation mehr aufgerufen werden können.
; Beim zweiten Aufruf wird die Atari-Taste reaktiviert.

CODE  segment
      assume cs:code,ds:code,es:code

start:

      mov ax,3509h
      int 21h             ;Flag holen
      mov di,offset flag
      mov ax,es:[di]
      cmp ax,11111        ;gesetzt, reinstallieren
      jz reinstall

; Programm installieren und resident im Speicher belassen
```

```
        mov cs:flag,11111  ;Flag setzen
        mov ax,2001h       ;Funktion 20h
        mov dl,0           ;Kein Zurücksetzen des Interruptvektors 9
        int 61h
        mov ax,cs
        sub ax,10h         ;Segmentadresse PSP merken
        mov cs:pspadr,ax
        mov ax,3509h       ;Interruptvektor 9 holen
        int 21h
        mov cs:int9_ofs,bx ;und sichern
        mov cs:int9_seg,es
        mov ax,2509h       ;Interruptvektor 9 neu setzen
        push cs            ;Segment = Code-Segment
        pop ds
        mov dx,offset int9
        int 21h
        mov ah,09          ;Meldung ausgeben
        mov dx,offset m1   ;Adresse des Strings
        int 21h
        mov ax,3100h       ;Programm resident beenden
        mov dx,big         ;Anzahl Paragraphen holen
        int 21h

; Programm reinstallieren und Speicher freigeben

reinstall:

        mov ax,2509h       ;Interruptvektor auf alte Routine
        mov dx,es:int9_ofs
        mov ds,es:int9_seg
        int 21h
        mov es,es:pspadr   ;Segmentadresse PSP holen
        mov ax,es
        mov es,es:[02ch]
        mov ah,49h         ;Environment-Speicher freigeben
        int 21h
        mov es,ax
        mov ah,49h         ;Programm-Speicher freigeben
        int 21h
        push cs
        pop ds
        mov ah,09          ;Meldung ausgeben
        mov dx,offset m2   ;Adresse des Strings
        int 21h
        mov ah,4ch         ;Programm beenden
        int 21h

; Der neue Interrupthandler 9

int9:   push ax
        push bx
        push dx
        push si
        push ds
```

```
        mov ax,0040h
        mov ds,ax
        mov dx,8000h
        in al,dx
        cmp al,80h                  ;Atari-Taste gedrückt,
        jz wan                      ;dann Code umwandeln
        cmp al,0
        jz wan
        jnz 14
wan:    mov al,155
14:     db  80h,0Eh,22h,01h,80h     ;or BYTE PTR [0122],80h
        mov bx,ds:[0151h]
        sub bx,ds:[0153h]
        ja 11
        add bx,0Ah
11:     cmp bx,04
        ja 12
13:     db 234,16h,15h,00,224       ;JMP E000:1516
12:     db 234,2dh,15h,00,224       ;JMP E000:152D

flag      dw 0
pspadr    dw 0
m1        db 13,'Atari-Taste wurde deaktiviert.',13,'$'
m2        db 13,'Atari-Taste wurde wieder aktiviert.',13,'$'
int9_ofs  dw 0
int9_seg  dw 0
big       EQU (($-start)/16)+11h   ;Größe des Programms in Paragraphen

CODE Ends
END
```

Nun folgt das Programmlisting für die BIOS-Version 1.072:

```
; Atari-DEAKTIVATOR BIOS 1.072
; written by Frank Riemenschneider
; Postfach 730309
; 3000 Hannover 71
; Beim ersten Aufruf wird die Atari-Taste deaktiviert,
; so daß keine Applikation mehr aufgerufen werden können.
; Beim zweiten Aufruf wird die Atari-Taste reaktiviert.

CODE  segment
      assume cs:code,ds:code,es:code

start:

        mov ax,3509h
        int 21h                     ;Flag holen
        mov di,offset flag
        mov ax,es:[di]
        cmp ax,11111                ;gesetzt, reinstallieren
        jz reinstall
```

```
; Programm installieren und resident im Speicher belassen

        mov cs:flag,11111       ;Flag setzen
        mov ax,2001h            ;Funktion 20h
        mov dl,0                ;Kein Zurücksetzen des Interruptvektors 9
        int 61h
        mov ax,cs
        sub ax,10h              ;Segmentadresse PSP merken
        mov cs:pspadr,ax
        mov ax,3509h            ;Interruptvektor 9 holen
        int 21h
        mov cs:int9_ofs,bx      ;und sichern
        mov cs:int9_seg,es
        mov ax,2509h            ;Interruptvektor 9 neu setzen
        push cs                 ;Segment = Code-Segment
        pop ds
        mov dx,offset int9
        int 21h
        mov ah,09               ;Meldung ausgeben
        mov dx,offset m1        ;Adresse des Strings
        int 21h
        mov ax,3100h            ;Programm resident beenden
        mov dx,big              ;Anzahl Paragraphen holen
        int 21h

; Programm reinstallieren und Speicher freigeben

reinstall:

        mov ax,2509h            ;Interruptvektor auf alte Routine
        mov dx,es:int9_ofs
        mov ds,es:int9_seg
        int 21h
        mov es,es:pspadr        ;Segmentadresse PSP holen
        mov ax,es
        mov es,es:[02ch]
        mov ah,49h              ;Environment-Speicher freigeben
        int 21h
        mov es,ax
        mov ah,49h              ;Programm-Speicher freigeben
        int 21h
        push cs
        pop ds
        mov ah,09               ;Meldung ausgeben
        mov dx,offset m2        ;Adresse des Strings
        int 21h
        mov ah,4ch              ;Programm beenden
        int 21h

; Der neue Interrupthandler 9

int9:   push ax
        push bx
```

```
        push dx
        push si
        push ds
        mov ax,0040h
        mov ds,ax
        mov dx,8000h
        in al,dx
        cmp al,80h                  ;Atari-Taste gedrückt,
        jz wan                      ;dann Code umwandeln
        cmp al,0
        jz wan
        jnz 14
wan:    mov al,155
14:     db  80h,0Eh,25h,01h,80h     ;or BYTE PTR [0125],80h
        mov bx,ds:[0153h]
        sub bx,ds:[0155h]
        ja 11
        add bx,0Ah
11:     cmp bx,04
        ja 12
13:     db 234,6Ah,15h,00,224       ;JMP E000:156A
12:     db 234,81h,15h,00,224       ;JMP E000:1581

flag       dw 0
pspadr     dw 0
m1         db 13,'Atari-Taste wurde deaktiviert.',13,'$'
m2         db 13,'Atari-Taste wurde wieder aktiviert.',13,'$'
int9_ofs   dw 0
int9_seg   dw 0
big        EQU (($-start)/16)+11h   ;Größe des Programms in Paragraphen

CODE Ends
END
```

2.8.3 Port-Capture: das Foto-Programm für den Portfolio

Ich möchte nun zu dem sicherlich schwierigsten Teil dieses Buches kommen, den »reinrassigen« TSR-Programmen, die per Tastendruck aktiviert werden können.

MS-DOS ist zwar das weitverbreitetste Betriebssystem aller Zeiten, fortschrittlich kann man es aber nicht gerade nennen. Durch schlecht programmierte Systemroutinen hilft bei immer größeren Programmen nur eine ständige Weiterentwicklung der Hardware, die z.Z. in dem 80486-Prozessor mit 33 MHZ Taktfrequenz gipfelt. Ein Problem kann aber auch ein noch so schneller Prozessor nicht beseitigen: MS-DOS ist ein reines Single-Task-System, das in keinster Weise auf Multitasking ausgelegt ist,

nicht einmal ein Task-Switching (mehrere Programme im Speicher, die wechselweise abgearbeitet werden) wird unterstützt, ganz im Gegenteil. Wer wie ich schon einmal auf einem Home-Computer des Typs *Amiga* gearbeitet hat, fragt sich, wieso ein ganzer Industriestandard immer noch hinter einem solchen Low-Cost-Rechner, wie es der *Amiga* nun einmal ist, hinterherhinkt.

Nun, diese Frage zu klären, wäre an dieser Stelle sicherlich unsinnig. Zumindest kann man sich aber darüber Gedanken machen, ob es nicht vielleicht doch möglich ist, einen Hauch von Task-Switching unter MS-DOS zu realisieren. Die erste Frage ist die, wie man ein zweites oder drittes Programm überhaupt während des Ablaufs eines Programms aktivieren könnte.

Da MS-DOS natürlich vom System keinerlei Unterstützung bietet, muß man sich etwas anderes einfallen lassen, nämlich eine Hardware-Lösung, da softwaremäßig keinerlei Chance besteht. Bevor man aber nun beginnt, eine Platine zu löten, sollte man sich die internen Möglichkeiten des Rechners ansehen: Hierbei fällt sofort der Tastatur-Interrupt $09 ins Auge. Bei jedem Tastendruck wird das laufende Programm unterbrochen. Damit hat man die Möglichkeit, die Kontrolle über das System zu übernehmen, ohne daß das ablaufende Programm etwas daran ändern könnte. Als »Startschuß« für die Aktivierung eines zweiten Programms könnte man so z.B. ungewöhnliche Tastenkombinationen wählen, die im Normalfall unsinnig sind (z.B. *beide* [Shift]-Tasten gedrückt). In dem Original-Handler des Interrupt $09 wird ein sogenanntes Tastatur-Status-Byte erzeugt, das Aufschluß insbesondere über die Tasten [Shift], [Alt] und [Strg] liefert. Dadurch bieten sich gerade diese Tasten zum Aufruf eines TSR-Programms an.

Zu beachten ist in diesem Zusammenhang, daß auf einen Aufruf des Original-Handlers auf keinen Fall verzichtet werden kann. Er stellt nämlich die Verbindung zur Hardware her, indem er den Tastaturprozessor ausliest, die Codes in ASCII-Zeichen umwandelt und diese in den Tastaturpuffer einträgt. Ohne den Original-Handler sind somit keine Eingaben über die Tastatur mehr möglich, was natürlich nicht im Sinne des Erfinders wäre. Ebenso besteht die Möglichkeit, daß unser TSR-Programm nicht das erste seiner Art ist: Würde es nicht den bislang gültigen Interrupt-Handler aufrufen, könnten alle vorher installierten TSR-Programme nicht mehr auf ihren Hot-Key reagieren. Man muß sich die Reihe der TSR-Programme wie eine Kette vorstellen: Das zuerst installierte TSR-Programm ruft den Original-BIOS-Handler auf, bevor die eigene

Routine abgearbeitet und auf einen Hot-Key geprüft wird. Das zweite Programm ruft zunächst den alten Handler, also den des ersten TSR-Programms auf, bevor dessen Rotuine abgearbeitet wird, usw. Aus diesem Grund (und einem weiteren, s.u.) dürfen zwei TSR-Programme nicht den gleichen Hot-Key benutzen.

Man muß jetzt noch überlegen, wie die alten Interrupt-Handler aufgerufen werden können: Über den INT-Befehl ist dies sicher nicht möglich, da dadurch ja immer wieder die gleiche Routine aufgerufen würde, was nach einer gewissen Anzahl von Aufrufen zum Stack-Überlauf und damit zum Absturz des Rechners führen würde. Man muß deshalb die Adresse des alten Handlers in eine Variable retten, damit die Routine über einen FAR-CALL-Befehl aufgerufen werden kann. Weil beim INT-Aufruf automatisch das Flag-Register auf den Stack gerettet wird, muß man dieses mit Hilfe eines PUSHF-Befehls vor dem FAR-CALL simulieren. Würde man dies vergessen, würde durch den IRET-Befehl am Ende des Interrupt-Handlers ein Flag-Register vom Stack geholt, das sich dort gar nicht befindet. Vielmehr würde hierfür ein Teil der Rücksprungadresse genommen, weshalb die Rücksprungadresse verfälscht wird und das Programm abstürzt.

Nachdem der alte Interrupt-Handler abgearbeitet wurde, kann dann an Hand des Tastatur-Flag, das man aus der BIOS-Variablenadresse $0040:$0017 auslesen kann, feststellen, ob ein Hot-Key gedrückt wurde, und das TSR-Programm aktivieren. Dies klingt zwar einfach, ein paar Hindernisse gibt es aber schon noch.

Das größte Problem besteht darin, daß DOS nicht *reentrant* ist. (Mit dem Begriff *reentrant* beschreibt man die Fähigkeit eines Betriebssystems, seine Funktionen von mehreren Programmen gleichzeitig ausführen zu lassen.)

Da das ablaufende Programm zu jeder Zeit per Hot-Key unterbrochen werden kann (Hardware-Interrupt!), kann man nie ausschließen, daß gerade eine DOS-Funktion ausgeführt und dann unterbrochen wurde. Dies ist nicht schlimm, solange das TSR-Programm nach seiner Beendigung korrekt in die unterbrochene DOS-Funktion zurückkehrt. Die Probleme treten erst dann auf, wenn das TSR-Programm seinerseits eine DOS-Funktion aufruft, was sowieso schon kaum zu verhindern ist, in Hochsprachen durch die Systemroutinen der Compiler jedoch zwangsläufig auftritt.

Um dies zu verstehen, muß man wissen, daß DOS beim Aufruf einer Funktion des Interrupt $21 die Register SS (Stacksegment) und SP

(Stackpointer) mit der Adresse eines der drei DOS-Stacks lädt. DOS benutzt also nicht den Stack des aufrufenden Programms, sondern einen eigenen. Auf diesem werden in erster Linie Rücksprungadressen beim Aufruf von Unterprogrammen abgelegt, er dient aber auch als Zwischenspeicher für interne Variablen. Das Problem liegt nun darin, daß man von außen nicht festlegen kann, welchen Stack DOS zu benutzen hat. Ruft man nun eine weitere DOS-Funktion aus einer unterbrochenen DOS-Funktion auf, kann man das Pech haben, daß die Funktion des TSR-Programms denselben Stack benutzt wie die unterbrochene Funktion. Damit werden natürlich alle Daten der unterbrochenen Funktion auf diesem Stack gelöscht. Zwar arbeitet die Funktion des TSR-Programms einwandfrei (solange sie nicht durch weitere TSR-Programme unterbrochen wird), nach der Rückkehr in die unterbrochene Funktion findet diese jedoch zerstörte Daten auf dem Stack vor, was meistens einem Systemabsturz gleichkommt.

Man darf also TSR-Aufrufe nur dann zulassen, wenn gerade keine DOS-Funktion ausgeführt wird. Zum Glück benutzt DOS ein Flag mit dem Namen *Indos*, das die Verschachtelungstiefe bei DOS-Aufrufen angibt. Enthält es den Wert Null, wird gerade keine DOS-Funktion ausgeführt, beim Wert 1 ist dies hingegen der Fall. Unter bestimmten Umständen darf DOS intern auch weitere DOS-Funktionen aufrufen, so daß Werte größer als 1 auch in seltenen Fällen möglich sind.

Natürlich darf das *Indos*-Flag vom TSR-Programm nicht mit Hilfe einer DOS-Funktion abgefragt werden, man kann den Wert jedoch auch direkt aus dem Speicher auslesen. Die Adresse kann man mit Hilfe der DOS-Funktion $34 bei der Initialisierung des TSR-Programms ermitteln, die die Segment- und Offsetadresse in ES:BX zurückliefert.

Der neue Handler für den Interrupt $09 muß nun also so ergänzt werden, daß er zwar weiterhin zunächst auf einen gedrückten Hot-Key testet, vor der tatsächlichen Aktivierung des TSR-Programms aber zunächst auch das *Indos*-Flag überprüft. Damit sind wir beim größten Problem überhaupt angekommen, dem Kommandoprozessor. Dieser benutzt nämlich ständig selbst DOS-Funktionen, was z.B. mit der Ausgabe des Prompt und dem Warten auf eine Tastatureingabe zusammenhängt. Das *Indos*-Flag hat somit permanent den Wert 1, weshalb man ein TSR-Programm nicht vom Kommandoprozessor aus aufrufen könnte. Mit dieser Einschränkung kann man nicht leben, weshalb auch hier nach einem Ausweg gesucht werden muß.

Beim PC wurde dies wie folgt realisiert: Um keine Zeit zu verschwenden, ruft COMMAND.COM in periodischen Abständen den Interrupt $28 auf, der z.B. für die Hintergrundausgabe mittels PRINT-Befehl verantwortlich ist. Während der Ausführung dieses Interrupt ist eine Unterbrechung zulässig, auch wenn das *Indos*-Flag einen Wert von ungleich Null aufweist.

Das Problem besteht beim Portfolio darin, daß hier vom Kommandoprozessor der Interrupt $28 nicht aufgerufen wird, so daß diese Methode scheitert: Genau deshalb laufen auch keine PC-TSR-Programme auf dem Portfolio! Nach langem Suchen bin ich darauf gestoßen, daß beim Portfolio dafür eine Unterbrechung während des BIOS-Interrupt $16, Funktion $00, zulässig ist, der vom Kommandoprozessor indirekt über DOS aufgerufen wird, wenn auf eine Eingabe gewartet wird. Wir müssen also noch einen neuen Handler für den BIOS-Interrupt $16 schreiben, der beim Aufruf ein Flag setzt, durch das angezeigt wird, daß ein Aufruf des TSR-Programms trotz eines *Indos*-Flag ungleich Null zulässig ist. Ist die BIOS-Routine beendet, wird das Flag wieder gelöscht.

Eine Einschränkung habe ich aus Vorsicht noch gemacht: Ist gerade ein BIOS-Interrupt $13 aktiv, durch den der direkte Zugriff auf ein Speichermedium vollzogen wird, wird die Aktivierung des TSR-Programms ebenfalls verhindert. Warum dies? Nun, beim PC sind Diskettenlaufwerke und Festplatten angeschlossen, die durch den Interrupt $13 gesteuert werden. Nehmen Sie an, auf der Platte soll ein Sektor beschrieben werden. Die Platte dreht sich, der Lesekopf ist auf Schreiben umgeschaltet, d.h., er magnetisiert die sich drehende Scheibe. Falls nun eine Unterbrechung erfolgt, dreht sich die Platte natürlich weiter und der Lesekopf magnetisiert, d.h., es wird nicht nur ein Sektor, sondern die ganze Spur mit unsinnigen Daten beschrieben. Beim Portfolio existieren keine derartigen zeitkritischen Aktionen, da hier ja nur auf RAM-Karten geschrieben wird. Tortzdem kann es passieren, daß ein Hersteller eines Tages ein Diskettenlaufwerk als Zubehör anbietet, womit die geschilderten Probleme auch auf den Portfolio zukämen. Da ich einen TSR-Handler schreiben wollte, der auch in 20 Jahren noch von Ihnen zu verwenden ist, habe ich das PC-Prinzip der Vorsicht schon integriert: Durch einen neuen Handler des Interrupt $13 wird ein Flag gesetzt, das eine Aktivierung des TSR-Programms bei der Ausführung des Interrupt verhindert.

Schließlich müssen noch rekursive Aufrufe des TSR-Programms aus sich selbst verhindert werden, was ebenfalls mit Hilfe eines Flag passieren kann.

Wir haben nun alle Voraussetzungen für die Aktivierung eines TSR-Programms besprochen. Was jetzt noch fehlt, ist der sogenannte Kontextwechsel zwischen den Programmen. Hiermit hat es folgendes auf sich: Damit das unterbrochende Programm nach dem Ende des TSR-Programms weiterlaufen kann, als sei nichts geschehen, müssen gewisse Informationen gerettet werden. Das sind zunächst alle Prozessorregister und der Stack. Daneben müssen aber auch alle betriebssystemabhängigen Komponenten wie die Adressen des PSP und der DTA gerettet werden, um nach der Rückkehr wieder auf die ursprünglichen Werte gesetzt werden zu können. Für die Ermittlung der Adressen und das Neusetzen existieren vier DOS-Funktionen. Falls das TSR-Programm einen eigenen Bildschirm aufbaut, müssen zusätzlich noch der Video-Modus und das Video-RAM gerettet werden. Da dies aber nicht immer der Fall ist (z.B. arbeiten Rechtschreibhilfen oft völlig im Hintergrund), kann man dies auch dem eigentlichen TSR-Programm überlassen.

Auf der anderen Seite benötigt das TSR-Programm vor seiner Aktivierung einen sinnvollen Inhalt der Prozessorregister und des Stack, mit dem es weiterarbeiten kann. Ebenso sind die Adressen des PSP und der DTA erforderlich. Diese Dinge müssen also schon bei der Initialisierung des TSR-Programms ermittelt und abgespeichert werden, damit die Werte bei dem Aufruf des TSR-Programms mittels Hot-Key wieder eingesetzt werden können.

Das folgende Programm stellt ein Capture-Programm für den Portfolio dar, also ein Programm, mit dem man den aktuellen Bildschirm abspeichern kann. Der TSR-Handler selbst ist zwangsläufig in Assembler geschrieben, das TSR-Programm muß jedoch in Pascal programmiert sein. Dem Handler wird bei der Installation einfach die Adresse einer Prozedur übergeben, die dann beim Drücken des Hot-Key aufgerufen wird *(TsrInit)*. Weiterhin müssen der Speicherbedarf des TSR-Programms in Paragraphen (16 Byte) und der Tastaturcode für den gewünschten Hot-Key übergeben werden. Hierfür gilt:

1: rechte [Shift]-Taste
2: linke [Shift]-Taste
4: [Strg]-Taste
8: [Alt]-Taste

Um z.B. eine Aktivierung des TSR-Programms bei einem Druck beider [Shift]-Tasten zu erreichen, muß der Wert 3 (1+2) übergeben werden. Bei der Speicherbestimmung gibt es beim Portfolio ein Problem. Normalerweise kann man in Turbo Pascal die Differenz der Variablen *FreePtr^*

(Spitze des Heap) und *PrefixSeg* (Startadresse des PSP) nehmen. Leider zeigt beim Portfolio wegen eines Fehlers in den Speicherverwaltungsroutinen die Variable *FreePtr^* immer auf die Spitze des verfügbaren RAM, egal wieviel Speicher Sie für den Heap mit der Compileranweisung {$M...} reserviert haben. Da man unmöglich das gesamte RAM für das TSR-Programm reservieren kann, muß man auf die Variable *HeapPtr^* zurückgreifen, die die Basis des Heaps angibt. Wenn man nun Daten auf dem Heap abgespeichert hat, muß man deren Platzbedarf zu der Differenz von *HeapPtr^* und *PrefixSeg* in Paragraphen hinzufügen, d.h. die Byteanzahl vorher durch 16 dividieren. Um ein Entfernen durch erneuten Aufruf des Programms zu ermöglichen, wurden die Routinen *TestInst* und *TsrUnInst* geschrieben, die testen, ob das Programm bereits resident installiert wurde (Ergebnis: TRUE) und es bei Bedarf dann wieder aus dem Speicher entfernen. Diese drei Assemblerroutinen befinden sich im Modul *Tsr.obj*, das durch die Compileranweisung {$L..} eingebunden wird.

Das Programm ermittelt selbständig, ob zur Zeit des Aufrufs der Text- oder Grafikmodus aktiv war. So wird der Textschirm als ASCII-File abgelegt, wobei man noch die Wahl zwischen dem 40- und 80-Zeichen-Bildschirm hat. Der Grafikbildschirm wird als *.PCX*-File gespeichert, so daß die Daten mit fast allen Grafik- und Textverarbeitungsprogrammen weiterverarbeitet bzw. ausgegeben werden können (DeluxePaint II Enhanced, Word 5.0 etc.). Da Sie im Grafikmodus nichts sehen können, wird die Arbeit des Programms akustisch untermalt. Steht am Ende ein hoher Ton (1200 Herz), ging bei der Speicherung alles glatt, bei einem Fehler ertönt ein tiefer Ton (700 Herz). Da Sie ebenfalls durch den Grafikmodus bedingt keine Eingaben bezüglich des Dateinamens machen können, wird hierfür der Text *Grafik* mit einer anschließenden Zahl festgelegt, die bei jeder gespeicherten Grafik um eins hochgezählt wird. Die Ausgabe erfolgt auf die interne RAM-Disk (C:). Im Textmodus sind Dateiname und Laufwerk frei eingebbar.

```
PROGRAM PortCapture;
{written by Frank Riemenscheider
              Postfach 730309
              3000 Hannover 71}

USES DOS, PORTCRT;

{$M 6000, 0, 0}
{$L a:tsr.obj}

VAR Buffer   : array[0..79] of word; {Bildschirmspeicher}
    help : word;
```

```
   grazahl : byte;      {Anzahl gespeicherte Grafiken}

PROCEDURE TsrInst(Prozoff : word; Taste : word; Speicher : word) ; external;
FUNCTION TestInst : boolean ; external ;
PROCEDURE TsrUnInst; external;

{—— SaveZeile: sichert die erste Bildschirmzeile ——}

PROCEDURE SaveZeile;
VAR spalte : byte;                  { die aktuelle bearbeitete Spalte }
BEGIN
  FOR spalte := 0 to 79 DO BEGIN
    Buffer[spalte] := Memw[$B000:2*spalte];
  END;
END;

{—— HolZeile: Restauriert erste Bildschirmzeile ——}

PROCEDURE HolZeile;
VAR spalte : byte;                  { die aktuelle bearbeitete Spalte }
reg : registers;
BEGIN
  FOR spalte := 0 to 79 DO BEGIN
    Memw[$B000:2*spalte] := Buffer[spalte];
  END;
  reg.ah := $12;        {Screen Refresh}
  Intr(97,reg);
END;

{——— TSR-Prozedur muß NEAR sein ———}

{$F-}
PROCEDURE Tsr;
VAR Spalte,i,j,k,
    Zeile,sploop,zloop  : byte;
    name : string;
    groesse : char;
    f : file;
    buf : array[0..82] of byte;
    flag : boolean;
    adresse,result : integer;
    reg : registers;
CONST
 header1 : array[0..18] of byte = (10,5,1,1,0,0,0,0,0,239,0,63,0,
                                   240,0,64,0,0,0,0);
 header2 : array[0..5] of byte = (1,1,30,0,1,0);
 sp : array[0..1] of byte = (39,79);
 ze : array[0..1] of byte = (7,24);
BEGIN
  reg.ah := $0E;
  reg.al := 0;         {Modus holen (Text/Grafik)}
  Intr(97,reg);
  IF (reg.dl and 128) = 0 THEN BEGIN
    SaveZeile;                         {Textmodus}
```

```
    Zeile := WhereY;
    Spalte := WhereX;
    GotoXY(0,0);
    write('xxxxxxxx Filename : <            > xxxxxxxx');
    GotoXY(21,0);
    readln(name);
    name := copy(name,1,12);
    IF Copy(name,length(name)-3,1) <> '.' THEN BEGIN
      name := copy(name,1,8);
      name := name + '.txt';
    END;
    GotoXY(0,0);
    write('x Format: <1> 40x8 <2> 80x25 <0> Ende x');
    Repeat
      Repeat
      Until Keypressed;
      groesse := ReadKey;
    Until ((groesse = '1') or (groesse = '2') or (groesse = '0'));
    HolZeile;
    IF groesse <> '0' THEN BEGIN
      flag := false;
      Assign(f,name);
      Rewrite(f,1);
      IF IOresult = 0 THEN BEGIN
        zloop := ze[ord(groesse)-49];
        sploop := sp[ord(groesse)-49];
        FOR i:= 0 to zloop DO BEGIN
          FOR j:= 0 to sploop DO BEGIN
            buf[j] := Mem[$B000:i*160+2*j];
          END;
          buf[sploop+1] := 13;
          buf[sploop+2] := 10;
          blockwrite(f,buf[0],sploop+3,result);
        END;
        close(f);
        IF ((IOresult =0) and (result = (sploop+3))) THEN flag := true;
      END;
      IF NOT flag THEN BEGIN
        GotoXY(0,0);
        write('xx Fehler beim Speichern der Daten! xx');
        groesse := ReadKey;
        HolZeile;
      END;
    END;
    GotoXY(Spalte,Zeile);
  END
  ELSE BEGIN                        {Grafikmodus}
    sound(1200,20);
    str(grazahl,name);
    name := 'Grafik'+name+'.pcx';
    Assign(f,name);
    Rewrite(f,1);
    IF IOresult = 0 THEN BEGIN
      blockwrite(f,header1,19,result);
```

```
      FOR i:= 0 to 44 DO BEGIN
         buf[i] := 255;
      END;
      blockwrite(f,buf[0],45,result);
      blockwrite(f,header2,6,result);
      blockwrite(f,buf[0],58,result);
      FOR i:= 0 to 63 DO BEGIN
        adresse := i*30;
        FOR j:= 0 to 29 DO BEGIN
          buf[j*2] := $C1;
          buf[j*2+1] := Mem[$B000:adresse+j];
        END;
        blockwrite(f,buf[0],60,result);
      END;
      close(f);
      IF ((IOresult =0) and (result = 60)) THEN BEGIN
        sound(1200,30);
        inc (grazahl);
      END
      ELSE BEGIN
        sound(700,30);
      END;
    END;
  END;
END;

{----------               HAUPTPROGRAMM               -------------}

BEGIN
  grazahl := 0;
  IF (TestInst) THEN        {Programm installiert?}
    BEGIN
      ClrScr;
      writeln('Port-Capture wurde reinstalliert.');
      TsrUnInst;            {Ja, Programm reinstallieren}
    END
  ELSE                      {Nein, Programm installieren}
    BEGIN
      ClrScr;
      writeln('xxxxxx P O R T  -  C A P T U R E xxxxxx');
      writeln('x Das Foto-Programm für den Portfolio x');
      writeln('x (C)opyright 1990 by Markt & Technik x');
      writeln('x   written by Frank Riemenschneider  x');
      writeln('x   Port-Capture wurde installiert.   x');
      writeln('x Aufruf mit <LEFTSHIFT>+<RIGHTSHIFT> x');
      write('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx');
      help := Seg(HeapPtr^)-PrefixSeg +1;   {Benötigter Speicherplatz}
      TsrInst(Ofs(Tsr),1+2,help);
    END;
END.
```

Hier nun der TSR-Handler, den Sie in alle von Ihnen entwickelten TSR-Programme für den Portfolio einbauen können:

```
; TSR-HANDLER
; written by
; Frank Riemenschneider
; Postfach 730309
; 3000 Hannover 71
; Beim ersten Aufruf wird der Handler installiert. Es stehen die Routinen
; TSRINST (Pointer auf TSR-Pascal Prozedur,Hotkeys),
; tpesTINST
; sowie TSRUNINST
; für einen Aufruf von Turbo Pascal zur Verfügung.
; Beim zweiten Aufruf wird der Handler wieder deaktiviert.


CODE        segment byte public

            assume cs:CODE, ds:CODE, es:CODE

public      tsrinst
public      testInst
public      tsruninst

;Aktivierung der neuen Interrupt-Routinen

tsrinst     proc    near

            push bp                 ;BP auf dem Stack sichern
            mov  bp,sp              ;SP nach BP übertragen
            push es                 ;ES auf dem Stack sichern

; Die Pascal-Programm-Segmentregister retten

            mov  cs:tpss,ss
            mov  cs:tpsp,sp
            mov  cs:tpes,es
            mov  cs:tpds,ds

            mov  bx,[bp+8]          ;Pointer auf TSR-Prozedur holen
            mov  cs:tsradr,bx       ;und sichern
            mov  bx,[bp+6]          ;Maske für Hotkey holen und merken
            mov  cs:hotkey,bx       ;und sichern

;PSP des Pascal-Programms sichern

            mov  ax,cs
            sub  ax,10h             ;256 Byte abziehen (10h Paragraphen)
            mov  cs:tppspseg,ax     ;Segmentadresse merken

;Adresse des INDOS-Flags sichern

            mov  ah,34h             ;Adresse INDOS-Flag holen
```

```
        int  21h
        mov  cs:indosofs,bx     ;und merken
        mov  cs:indosseg,es

;DTA-Adresse des Pascal-Programms merken

        mov  ah,2fh             ;DTA-Adresse holen
        int  21h
        mov  cs:tpdtaofs,bx
        mov  cs:tpdtaseg,es     ;und merken

;Installationsflag (TSR bereits installiert) setzen

        mov  ax,11111
        mov  cs:aktflag,ax

;Adressen der umzuleitenden Interrupt-Handler merken

        mov  ax,3509h           ;Interrupt-Vektor 9h sichern
        int  21h
        mov  cs:int9ofs,bx
        mov  cs:int9seg,es

        mov  ax,3513h           ;Interrupt-Vektor 13h sichern
        int  21h
        mov  cs:int13ofs,bx
        mov  cs:int13seg,es

        mov  ax,3516h           ;Interrupt-Vektor 16h sichern
        int  21h
        mov  cs:int16ofs,bx
        mov  cs:int16seg,es

;Zurücksetzen des Interrupt-Vektors 9h verhindern

        mov ah,20h
        mov al,1
        mov dl,0
        int 61h

;Installation der Interrupt-Vektoren

        push ds                 ;Datensegment merken
        mov  ax,cs              ;Segment ist immer Codesegment
        mov  ds,ax

        mov  ax,2509h           ;Interrupt-Vektor 9h setzen
        mov  dx,offset int09
        int  21h

        mov  ax,2513h           ;Interrupt-Vektor 13h setzen
        mov  dx,offset int13
        int  21h
```

```
        mov  ax,2516h          ;Interrupt-Vektor 16h setzen
        mov  dx,offset int16
        int  21h

        pop  ds                ;DS wieder vom Stack holen

;Programm resident beenden

        mov  ax,3100h
        mov  dx,[bp+4]         ;Anzahl der reservierten Paragraphen
        int  21h               ;Programm beenden

tsrinst  endp

; testINST: stellt fest, ob das Programm bereits installiert ist
; Rückgabe-Wert : 1, wenn das Programm installiert ist, sonst 0

testinst  proc   near

        mov  ax,3509h          ;Interrupt-Vektor 9h holen
        int  21h               ;DOS-Interrupt holt Segadresse nach ES
        mov  di,offset aktflag ;Installationsflag holen
        mov  ax,11111          ;Vergleichswert holen

        mov dl,0               ;Annahme: Nicht identisch
        cmp  ax,es:[di]        ;mit Installationsflag vergleichen
        jne  notinst           ;ungleich: Nicht installiert

        mov  dl,1              ;gleich: Installiert
notinst: mov  al,dl            ;Return-Code nach AL holen
        ret                    ;zurück zum Aufrufer

testinst  endp                 ;Ende der Prozedur

; TSRUNINST: reinstalliert das TSR-Programm

tsruninst proc   near

        push ds                ;DS auf dem Stack sichern

        mov ax,3509h           ;Holt Segmentadresse der neuen
        int 21h                ;Interrupt-Routinen

        cli

        mov  ax,2509h          ;Alten Interrupt-Vektor 9h setzen
        mov  ds,es:int9seg
        mov  dx,es:int9ofs
        int  21h

        mov  ax,2513h          ;Alten Interrupt-Vektor 13h setzen
        mov  ds,es:int13seg
        mov  dx,es:int13ofs
        int  21h
```

```
        mov  ax,2516h           ;Alten Interrupt-Vektor 16h setzen
        mov  ds,es:int16seg
        mov  dx,es:int16ofs
        int  21h

        sti                     ;Interrupts wieder erlauben

        mov  es,es:tppspseg     ;PSP-Segmentadresse des TSR-Programms
        mov  cx,es              ;holen
        mov  es,es:[02ch]       ;Segmentadresse Environment aus PSP holen
        mov  ah,49h
        int  21h                ;Speicher freigeben

        mov  es,cx
        mov  ah,49h
        int  21h                ;Speicher freigeben

        pop  ds                 ;DS wieder vom Stack holen
        ret

tsruninst endp                  ;Ende der Prozedur

;Die neue Interrupt 09h-Routine

int09   proc far

        pushf                   ;Aufruf der alten Routine
        call cs:int9ptr

        cli                     ;Interrupts verbieten
        cmp  cs:rekflag,0       ;TSR-Programm bereits aktiv?
        jne  notsr              ;Ja, Ende!

        cmp  cs:int13flag,0     ;BIOS-Disk-Interrupt aktiv?
        jne  notsr              ;Ja, Ende!

        push ax
        push es
        xor ax,ax               ;Hot-Key betätigt?
        mov es,ax
        mov ax,word ptr es:[417h]
        and ax,cs:hotkey
        cmp ax,cs:hotkey
        pop es
        pop ax
        jne  notsr              ;Nein, Ende!

        push ds
        push bx
        lds  bx,cs:indos
        cmp  byte ptr [bx],0    ;INDOS-Flag holen
        pop  bx
        pop  ds                 ;DOS-Funktion aktiv?
```

```
           je  tsrstart            ;Nein, TSR-Aufruf möglich!
           push ax
           mov ah,cs:int16flag     ;BIOS-Flag prüfen
           cmp ah,0                ;= 0, dann warten auf Tastendruck
           pop ax
           jnz notsr               ;Nein, Ende!

tsrstart:  call tsrbegin           ;TSR-Programm aufrufen

notsr:     sti                     ;Interrupts wieder zulassen
           iret                    ;zurück zum unterbrochenen Programm

int09      endp


;TSR-Programm aktivieren

tsrbegin   proc near

           mov  cs:rekflag,1       ;TSR-Rekursions-Flag setzen

           mov  cs:stseg,ss        ;aktuelles Stacksegment und Stack-
           mov  cs:stptr,sp        ;zeiger merken

           mov  ss,cs:tpss         ;den Stack des Pascal-Programms
           mov  sp,cs:tpsp         ;aktivieren

           push es                 ;die Register auf dem Stack
           push ds                 ;des Pascal-Programms sichern
           push di
           push si
           push bp
           push dx
           push cx
           push bx
           push ax

           mov cx,64
           mov ds,cs:stseg
           mov si,cs:stptr        ;DOS-Stack sichern

lab1:      push word ptr [si]
           inc si
           inc si
           loop lab1

           mov  ah,51h            ;Adresse des PSP holen
           int  21h
           mov  cs:pspseg,bx      ;und merken

           mov  ah,2fh            ;DTA-Adresse holen
           int  21h
           mov  cs:dtaofs,bx
           mov  cs:dtaseg,es      ;und merken
```

```
            mov  ah,50h           ;PSP des Pascal-Programms setzen
            mov  bx,cs:tppspseg
            int  21h

            mov  ah,1ah           ;DTA-Adresse des Pascal-Programms setzen
            mov  dx,cs:tpdtaofs
            mov  ds,cs:tpdtaseg
            int  21h

            mov  ds,cs:tpds       ;Segment-Register für das Pascal-
            mov  es,cs:tpes       ;Programm setzen

            sti                   ;Interrupts wieder zulassen
            call cs:tsradr        ;TSR-Programm aufrufen
            cli                   ;Interrupts verbieten

            mov  ah,1ah           ;DTA-Adresse des unterbrochenen
            mov  dx,cs:dtaofs
            mov  ds,cs:dtaseg
            int  21h              ;Programms setzen

            mov  ah,50h           ;PSP des unterbrochenen
            mov  bx,cs:pspseg
            int  21h              ;Programms setzen

            mov cx,64
            mov ds,cs:stseg       ;DOS-Stack vom Stack des
            mov si,cs:stptr
            add si,128            ;Pascal-Programms holen
lab2:       dec si
            dec si
            pop word ptr [si]
            loop lab2

            pop  ax               ;Register vom
            pop  bx               ;Stack des Pascal-Programms holen
            pop  cx
            pop  dx
            pop  bp
            pop  si
            pop  di
            pop  ds
            pop  es

            mov  ss,cs:stseg      ;Stack des unterbrochenen Programms
            mov  sp,cs:stptr      ;restaurieren

            mov  cs:rekflag,0     ;TSR-Rekursions-Flag zurücksetzen
            ret

tsrbegin    endp
```

```
;Die neue Interrupt 13h-Routine

int13       proc far

            mov   cs:int13flag,1     ;BIOS-Disk-Interrupt aktiv
            pushf                    ;Aufruf der alten Interrupt 13h-Routine
            call  cs:int13ptr
            mov   cs:int13flag, 0    ;BIOS-Disk-Interrupt nicht mehr aktiv
            ret   2                  ;zurück zum unterbrochenen Programm

int13       endp

;Die neue Interrupt 16h-Routine

int16       proc far
            mov   cs:int16flag,ah    ;BIOS-Funktionsnummer merken
            pushf                    ;Aufruf der alten Interrupt 16h-Routine
            call  cs:int16ptr
            mov   cs:int16flag,111   ;BIOS-Interrupt nicht mehr aktiv
            ret 2                    ;zurück zum unterbrochenen Programm
int16       endp

; Variablen

aktflag     dw 0                     ;Aktivierungs-Flag
tpss        dw 0                     ;Stacksegment des Pascal-Programms
tpsp        dw 0                     ;Stackpointer des Pascal-Programms
tpds        dw 0                     ;Datensegment des Pascal-Programms
tpes        dw 0                     ;Extrasegment des Pascal-Programms
tpdtaofs    dw 0                     ;DTA-Adresse des Pascal-Programms
tpdtaseg    dw 0
tppspseg    dw 0                     ;Segmentadresse des PSP des Pascal-Prog.
tsradr      dw 0                     ;Adresse der TSR-Prozedur
hotkey      dw 0                     ;Hotkey-Maske für BIOS-Tastaturflag
rekflag     db 0                     ;Rekursionsflag
int13flag   db 0                     ;BIOS-Disk-Interrupt-Flag
int16flag   db 111                   ;BIOS-Tastatur-Interrupt-Flag
indos       equ this dword           ;Pointer auf das DOS-Indos-Flag
indosofs    dw 0                     ;INDOS-Flag Offsetadresse
indosseg    dw 0                     ;INDOS-Flag Segmentadresse
int13ptr    equ this dword           ;Alter Interrupt-Vektor 13h
int13ofs    dw 0                     ;Interrupt-Vektor 13h Offsetadresse
int13seg    dw 0                     ;Interrupt-Vektor 13h Segmentadresse
int16ptr    equ this dword           ;Alter Interrupt-Vektor 16h
int16ofs    dw 0                     ;Interrupt-Vektor 16h Offsetadresse
int16seg    dw 0                     ;Interrupt-Vektor 16h Segmentadresse
int9ptr     equ this dword           ;Alter Interrupt-Vektor 9h
int9ofs     dw 0                     ;Interrupt-Vektor 9h Offsetadresse
int9seg     dw 0                     ;Interrupt-Vektor 9h Segmentadresse
dtaofs      dw 0                     ;DTA-Adresse (Offset)
dtaseg      dw 0                     ;DTA-Adresse (Segment)
pspseg      dw 0                     ;Segmentadresse des PSP
stseg       dw 0                     ;Stacksegment des unterbrochenen Prog.
stptr       dw 0                     ;Stackpointer des unterbrochenen Prog.

CODE        ends                     ;Ende des Codesegments
            end
```

2.9 Die Nachlade-Unit für den Portfolio

Zum Schluß dieses Kapitels möchte ich Ihnen sozusagen als Ausklang nach dem TSR-Streß eine Unit vorstellen, die es erlaubt, Programme nachzuladen, wobei Variablen übergeben werden können.

Turbo Pascal bietet zwar eine gute Overlay-Unit an, die jedoch den Nachteil aufweist, daß immer ein Modul im Speicher gehalten werden muß. Bei den Platzproblemen des Portfolio scheint es aber angebracht zu sein, auch das Nachladen eines Programmteils zu ermöglichen, wobei der erste komplett aus dem Speicher entfernt wird.

Der folgende Trick stammt aus meinen C64-Zeiten, wo ebenfalls massive Speicherprobleme auftraten. Er funktioniert, indem in den Tastaturpuffer per Hand die Codes eingetragen werden, die sonst von dem Interrupt-$09-Handler vorgenommen werden. Wenn Sie im Kommandoprozessor einen Programmnamen mit einem abschließenden Carriage-Return eingeben, bedeutet dies, daß dieser das Programm sucht und startet. Wenn Sie nun die einzelnen Codes des Programmnamens und den Code für den Carriage-Return ($13) bereits in einem anderen Programm in den Tastaturpuffer schreiben, müssen Sie hierzu keine Tasten mehr drücken: Sofort nach der Beendigung des Programms wird das nächste gestartet, also de facto nachgeladen. Die Variablen müssen jedoch vorher in eine Datei gerettet werden, da der Speicher des aufrufenden Programms nach dessen Ende freigegeben wird.

Die Unit besteht aus nur zwei Prozeduren: *Speichern* sichert die Variablen und lädt das folgende Programm nach. Dafür sind folgende Daten erforderlich:

- Name der ersten Variablen
- Name der letzten Variablen
- Größe der letzten Variablen in Byte
- Pfad und Name des nachzuladenden Programms
- Pfad und Name der Sicherungsdatei

Die Prozedur *Laden* benötigt nur den Namen der ersten Variablen und den Pfad- und Dateinamen der Sicherungsdatei.

Wichtig: Das aufrufende und das nachzuladende Programm müssen exakt die gleichen Variablen in der gleichen Reihenfolge deklariert haben, sonst funktioniert die Übergabe nicht!

```
UNIT portovr;
{written by Frank Riemenschneider
            Postfach 730309
            3000 Hannover 71}
INTERFACE
FUNCTION Speichern(VAR erste; VAR letzte; platz : word; name,datei : string) : boolean;
FUNCTION Laden(VAR erste; datei : string) : boolean;

IMPLEMENTATION

USES dos;

FUNCTION Speichern(VAR erste; VAR letzte; platz : word; name,datei : string) : boolean;
CONST
scan : array[0..64] of byte = ($39,$02,$03,$2B,$05,$06,$07,$0D,$09,$0A,
                               $1B,$1B,$33,$35,$34,$08,$0B,$02,$03,$04,
                               $05,$06,$07,$08,$09,$0A,$34,$33,$29,$0B,
                               $29,$0C,$00,$1E,$30,$2E,$20,$12,$21,$22,
                               $23,$17,$24,$25,$26,$32,$31,$18,$19,$10,
                               $13,$1F,$14,$16,$2F,$11,$2D,$2C,$15,$09,
                               $0C,$0A,$00,$35,$0D);
VAR
startof,endof,anzahl,zahl,i: word;
f : file;
help : byte;
flag : boolean;
BEGIN
 flag := false;
 startof := ofs(erste);
 endof := ofs(letzte);
 zahl := (endof-startof)+platz;
 assign(f,datei);
 rewrite(f,1);
 IF IOResult = 0 THEN BEGIN
   blockwrite(f,zahl,2,anzahl);
   blockwrite(f,erste,zahl,anzahl);
   close(f);
   IF ((anzahl = zahl) and (IOResult = 0)) THEN BEGIN
      flag := true;
      name := Copy(name,1,15);
      FOR i:= 1 to length(name) DO BEGIN
         name[i] := Upcase(name[i]);
         help := Ord(name[i]);
         mem[$0040:$001C+2*i] := help;
         mem[$0040:$001D+2*i] := scan[help-32];
      END;
      help := 2*length(name);
      memw[$0040:$001E+help] := 13;
      memw[$0040:$001F+help] := $1C;
      memw[$0040:$001A] := $1E;
      memw[$0040:$001C] := $1E+help+2;
   END;
```

```
 END;
 Speichern := flag;
END;

FUNCTION Laden(VAR erste; datei : string) : boolean;
VAR
anzahl,zahl : word;
f : file;
flag : boolean;
BEGIN
 flag := false;
 assign(f,datei);
 reset(f,1);
 blockread(f,zahl,2,anzahl);
 blockread(f,erste,zahl,anzahl);
 close(f);
 IF ((IOResult = 0) and (zahl = anzahl)) THEN flag := true;
END;

BEGIN
END.
```

Hier sind nun zwei Beispielprogramme, wobei das erste *(Test0)* das zweite *(Test1)* nachlädt.

```
PROGRAM TEST0;

{OVERLAy-DEMO
 written by Frank Riemenschneider
            Postfach 730309
            3000 Hannover 71}

{Das Programm füllt per Zufallsgenerator ein Record
 des Typs Adresse mit ASCII-Zeichen und lädt dann
 ein Programm "Test1" nach, welches auf diesen
 Record zugreifen wird, ohne ihn vorher initialisiert
 zu haben.}


USES dos,portovr;

TYPE adresse = record
      vorname: string[10];
      nachname: string[10];
      strasse : string[10];
      plz : word;
      ort : string[10];
      telefon : longint;
END;

CONST
    zahl : byte = 10;
```

```
    meldung = 'Overlay-Fehler! Bitte Taste drücken.';

VAR i,j : word;
    data : array[1..10] of adresse;
    f : boolean;
BEGIN
    Randomize;
    FOR i:= 1 TO zahl DO BEGIN
      data[i].vorname := '';
      data[i].nachname := '';
      data[i].ort := '';
      data[i].strasse := '';
      FOR j:= 1 TO zahl DO BEGIN
        data[i].vorname := data[i].vorname + chr(65+random(26));
        data[i].nachname := data[i].nachname + chr(65+random(26));
        data[i].ort := data[i].ort + chr(65+random(26));
        data[i].strasse := data[i].strasse + chr(65+random(26));
      END;
      data[i].telefon := Round(1000*Random(4));
      data[i].plz := Round(1000*Random(4));
    END;
    FOR i:= 1 TO zahl DO BEGIN
      writeln(data[i].vorname,' ',data[i].nachname);
      writeln(data[i].plz,' ',data[i].ort);
      writeln(data[i].strasse,' Telefon : ',data[i].telefon);
      writeln;
    END;
    f := speichern(i,f,sizeof(f),'a:\test1','c:\data');
    if NOT f then BEGIN
      writeln(meldung);
      readln;
      halt;
    END;
END.→

PROGRAM TEST1;

USES dos,portovr;

TYPE adresse = record
      vorname: string[10];
      nachname: string[10];
      strasse : string[10];
      plz : word;
      ort : string[10];
      telefon : longint;
END;
CONST
    zahl : byte = 10;
    tel : string[10] = 'Telefon : ';

VAR i,j : word;
    data : array[1..10] of adresse;
    f : boolean;
```

```
BEGIN
    f := laden(i,'c:\data');
    if NOT f then BEGIN
      writeln('Overlay-Fehler! Bitte Taste drücken.');
      halt;
    END;
    FOR i:= 1 TO zahl DO BEGIN
      writeln(data[i].vorname,' ',data[i].nachname);
      writeln(data[i].plz,' ',data[i].ort);
      writeln(data[i].strasse,' ',tel,data[i].telefon);
      writeln;
    END;
END.
```

2.10 Objektorientierte Programmierung auf dem Portfolio

Die objektorientierte Programmierung, auch kurz OOP genannt, ist als ein neues Konzept zur Lösung komplexer Programmieraufgaben zu verstehen. Dabei tun sich viele Leute beim Verstehen der OOP schwer; der beste Weg, in die Materie einzusteigen, ist, sich ein OOP-Programm zu nehmen und das Programm zu debuggen. Dieses wollen wir hier dann auch tun... Sie sollten dabei nicht gleich die Flinte ins Korn werfen, wenn Sie das Programm nicht gleich verstehen. Auch Turbo-Pascal-Profis haben zuweilen Probleme mit der objektorientierten Programmierung. Das Problem besteht meiner Ansicht nach vor allen Dingen darin, daß viele selbsternannte Experten, z.B. Redakteure von Fachzeitschriften, beliebig viele Artikel über OOP verfassen, jedoch offensichtlich ohne verstanden zu haben, worum es wirklich geht. Nur so kann ich mir erklären, daß seitenweise theoretische Abhandlungen vorgenommen werden, die aus irgendwelchen Büchern extrahiert wurden, wenn es aber darum geht, ein Beispielprogramm zu präsentieren, müssen die Herren allesamt passen. Ich jedenfalls habe bis heute noch in keinem Buch und in keiner Zeitschrift ein OOP-Programm gefunden, das halbwegs alle Möglichkeiten der OOP verdeutlicht hätte. Da man natürlich aber auch auf dem Portfolio OOP-Programme ablaufen lassen kann, möchte ich die Gelegenheit wahrnehmen, Ihnen ein Beispielprogramm zu präsentieren.

Um objektorientierte Programme verstehen zu können, brauchen Sie erst einmal eine Erläuterung der folgenden Begriffe:

Vererbung: Bei der Definition eines Objekts wird dieses in einen sog. Objektstammbaum eingeordnet. Jedes Objekt erbt dabei den Code und die Daten der Vorfahren.

Kapselung: In einem Objekt werden Code und Daten miteinander verbunden. Diese Verbindung bezeichnet man als Kapselung.

Polymorphie: Wird in einem Objekt eine Funktion definiert, dann implementiert jedes Objekt, das diese Funktion geerbt hat, die Funktion in der für dieses Objekt erforderlichen Art und Weise.

Methode: Eine in einem Objekt definierte Prozedur oder Funktion.

Das Programm SCHUELER.EXE dient zur Erfassung z.B. von Laufzeiten bei Sportveranstaltungen verschiedener Schulklassen. Dabei kann eine Liste sortiert nach Klassen und eine Liste sortiert nach Leistungen ausgegeben werden. Weiterhin können die Daten gespeichert und wieder geladen werden. Nach dem Start des Programms erscheint ein Menü, aus dem Sie durch Eingabe einer Ziffer die entsprechende Funktion auswählen können. Der Source-Code zu diesem Programm wird im folgenden dargestellt.

```
PROGRAM Schule;

{$M 10000,20000,20000}

{Beispiel für die Verwaltung einer Klassenstufe

written by Frank Riemenschneider
           Postfach 730309
           3000 Hannover 71}

USES dos,portcrt;

(xxxxxxxxxxxxxxxxxxxxxxxxxxxx Konstanten xxxxxxxxxxxxxxxxxxxxxxxxxxxx)

CONST
  zeit_8   =  8.30;          { Normalzeit Klasse 8 - hier als Beispiel }
  bon_8    =  30;            { Vorgabe der Mä. Klasse 8 - ebenso }
  tab      :  array[1..7] of byte = (1, 4, 16, 27, 31, 33, 35);
  maxzeile      = 23;
  anfangszeile = 4;          { ..der Bildschirmausgabe ohne Überschrift }
  fullstern : string[80] =
'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
```

```
  randstern : string[1] = 'x';
  ASCII = 0;
  Scan = 1;


(xxxxxxxxxxxxxxxxxxxxxxxxxxxxx Typ Deklarationen xxxxxxxxxxxxxxxxxxx)

TYPE
     Schuelerdaten = RECORD             { zur Datensicherung }
        Klassenbezeichnung              : string[7];
        Klassennummer                   : byte;
        Nummer                          : byte;
        Nachname                        : string[18];
        Vorname                         : string[18];
        genus                           : boolean;
        angemeldet                      : boolean;
        teilgenommen                    : boolean;
        Laufzeit                        : real;
     END;


(xxxxxxxxxxxxxxxxxxxxxxxxxxObjekttyp Deklarationen xxxxxxxxxxxxxxxxxxx)

TYPE
     Name = OBJECT
        nachname, vorname               : string[18];
        zeile, Xn, Xv                   : byte;
        CONSTRUCTOR init(nachn, vorn: string);
        FUNCTION getnachn: string;
        FUNCTION getvorn: string;
        FUNCTION getzeile: byte;
        PROCEDURE putzeile(zl: integer);
        PROCEDURE anlegen(zl: byte); virtual;
        FUNCTION z: byte;               { diese beiden Funktionen ermitteln...}
        FUNCTION seite: byte;           {...auf welcher Bildschirmhälfte... }
        PROCEDURE Ausgabe;virtual;      {...die Ausgabe erfolgen soll }
     END;


     Person = OBJECT(Name)
        genus                           : boolean;         { Mädchen/Junge }
        Xg                              : byte;
        CONSTRUCTOR init(nachn, vorn: string; gen: boolean);
        FUNCTION getgen:boolean;
        PROCEDURE anlegen(zl: byte); virtual;
        PROCEDURE ausgabe; virtual;
     END;

     Teilnehmer = OBJECT(Person)
        angemeldet, teilgenommen        : boolean;
        Laufzeit                        : real;
        Xa, Xt, Xl                      : byte;
        CONSTRUCTOR init(nachn, vorn: string; gen: boolean;
                         ang, tlg: boolean; lz: real);
```

```
    FUNCTION getang                     : boolean;
    FUNCTION gettlg                     : boolean;
    FUNCTION getlz                      : real;
    PROCEDURE tg;
    PROCEDURE ntg;
    PROCEDURE ummelden;
    PROCEDURE ausgabe; virtual;
    PROCEDURE lieszeit;
 END;

      scPtr    = ^Schueler;
      knPtr    = ^knoten;
 knoten   = RECORD
    next            : knPtr;
    item            : scPtr;
 END;


 Schueler     = OBJECT(Teilnehmer)
    Klassennummer, Xkl          : byte;
    nummer, Xnr                 : byte;
    Klassenbezeichnung          : string[7];
    Zkbz, Xkbz                  : byte;
    CONSTRUCTOR init(nachn, vorn, klbz: string; bon, ang, tlg: boolean;
                     lz: real; nr, klnr: byte);
    FUNCTION getnr: byte;
    FUNCTION getklnr: byte;
    FUNCTION getklbz: string;
    DESTRUCTOR done; virtual;
    PROCEDURE ausgabe; virtual;
    PROCEDURE ausgabe2; virtual;  {Ausgabe für die Jahrg. - Bestenl.}
    PROCEDURE anlegen(zl: byte); virtual;
    PROCEDURE maske;
 END;

 Jahrgangsliste = OBJECT
    CONSTRUCTOR init;
    PROCEDURE load;
    PROCEDURE store;
    PROCEDURE add(VAR K: knPtr; S: scPtr);
    PROCEDURE sortieren(VAR K: knPtr);
    PROCEDURE loeschen(klnr, nr: byte);
    PROCEDURE meldum(klnr, nr: byte);
    PROCEDURE ausgeben;    { wird noch nicht verwendet }
    DESTRUCTOR done; virtual;
 END;

 Jahrgangsbestenliste = OBJECT(Jahrgangsliste)
    start          : knPtr;
    CONSTRUCTOR init;
    PROCEDURE einsammeln(gen: boolean);
    PROCEDURE sortieren(VAR K: knPtr);
    PROCEDURE ausgeben;
```

```
      PROCEDURE herstellen;
      DESTRUCTOR done; virtual;
   END;


   Klassenbestenliste = OBJECT(Jahrgangsbestenliste)
      CONSTRUCTOR init;
      PROCEDURE einsammeln(gen: boolean; klnr: byte);
      PROCEDURE herstellen(klnr: byte);
   END;


   Klassenliste   = OBJECT(Jahrgangsliste)
      anfang                   : knPtr;
      CONSTRUCTOR init;
      FUNCTION ausgeben(klnr: byte): knPtr;
      PROCEDURE listeanlegen(klnr: byte);
      PROCEDURE numerieren(klnr: byte);
      PROCEDURE loeschen(klnr: byte);
      PROCEDURE ummelden(klnr: byte);
      PROCEDURE zeiteingeben(klnr: byte);
      FUNCTION durchschnitt(klnr: byte): string;
   END;


(xxxxxxxxxxxxxxxxxxxxxxx Variablen xxxxxxxxxxxxxxxxxxxxxxxx)

VAR
   modus                       : byte;
   wahl                        : string;
   SCR                         : knPtr;
   temp                        : string;
   root                        : knPtr;
   Einejahrgangsliste          : Jahrgangsliste;
   Einejahrgangsbestenliste    : Jahrgangsbestenliste;
   Eineklassenbestenliste      : Klassenbestenliste;
   Eineklassenliste            : Klassenliste;


{xxxxxxxxx Methoden des Typs "Name" xxxxxxxxxxxxxxxxxxx}

CONSTRUCTOR Name.init(nachn, vorn: string);
BEGIN
  nachname := nachn;
  vorname  := vorn;
  zeile    := anfangszeile;
  Xn       := tab[2] ;
  Xv       := tab[3] ;
END;


FUNCTION Name.getnachn: string;
BEGIN
  getnachn := nachname;
```

```
END;


FUNCTION Name.getvorn: string;
BEGIN
  getvorn := vorname;
END;


FUNCTION Name.getzeile: byte;
BEGIN
  getzeile := zeile;
END;


PROCEDURE Name.putzeile(zl: integer);
BEGIN
  zeile := zl;
END;


FUNCTION Name.seite: byte;
BEGIN
  IF zeile < maxzeile THEN seite := 0
  ELSE seite := 40;
END;


FUNCTION Name.z: byte;
BEGIN
  IF zeile < maxzeile THEN z := zeile
  ELSE z := zeile - maxzeile + anfangszeile;
END;


FUNCTION LesTaste(modus: byte) : byte;
VAR reg : registers;
BEGIN
   IF modus = 0 THEN BEGIN
     reg.ah := 0;
     intr(22,reg);
     LesTaste := reg.al;
   END
   ELSE BEGIN
     LesTaste := reg.ah;
   END;
END;


PROCEDURE Eingabe(var ss: string; x,y,maxlaenge,
                  numerisch : byte; vs: string);
VAR taste : byte;
posi,code,laenge : byte;
i : byte;
```

```
flag : boolean;
BEGIN
  GotoXY(x,y);
  FOR i:= 1 TO maxlaenge DO begin
    write(' ');
  end;
  GotoXY(x,y);
  write(vs);
  GotoXY(x,y);
  ss:= vs;
  laenge := length(ss);
  posi := 0;
  flag := true;
  REPEAT
    GotoXY(x+posi,y);
    taste := Ord(LesTaste(ASCII));                {ASCII-Code holen}
    code := Ord(LesTaste(scan));
    IF flag THEN begin
      IF ((code <> $1C) and (code <> $0E)
          and (code <> $52) and (code <> $53)
          and (code <> $4B) and (code <> $4D)
          and (code <> $47) and (code <> $4F)) then begin
        FOR i:= 1 TO maxlaenge DO begin
          write(' ');
        END;
        ss := '';
        laenge := 0;
        GotoXY(x,y);
      END;
    END;
    flag := false;

   IF (((laenge < maxlaenge) or (posi < laenge))
      and ((taste >31) and (taste<156))
      and ((numerisch<>0) or ((taste = 43) or
      (taste = 46) or ((taste >47) and (taste<58)))))
      THEN BEGIN
             IF posi = laenge THEN BEGIN
                 inc(laenge);
                 inc(ss[0]);
             END;
             inc(posi);
             write(chr(taste));
             ss[posi] := chr(taste);
   END;


    {Hier beginnen, wenn Backspace gedrückt. (Über Scan-Code)}

    CASE code OF


       $0E : BEGIN
              IF posi > 0 THEN BEGIN
```

```
            dec(posi);
            dec(laenge);
            ss := Copy(ss,1,posi)+Copy(ss,posi+2,100);
            GotoXY(x,y);
            write(ss+' ');
           END;
         END;

      $53 : BEGIN
           IF posi < laenge THEN BEGIN
            dec(laenge);
            ss := Copy(ss,1,posi)+Copy(ss,posi+2,100);
            GotoXY(x,y);
            write(ss+' ');
           END;
         END;

      $4B : IF posi > 0 THEN dec(posi);
      $4D : IF posi < laenge THEN inc(posi);

      $52 : BEGIN
           IF ((laenge < maxlaenge) and (posi < laenge)) THEN BEGIN
            ss := Copy(ss,1,posi)+' '+Copy(ss,posi+1,100);
            GotoXY(x,y);
            write(ss);
            inc(laenge);
           END;
         END;

      $47 : posi := 0;
      $4F : posi := laenge;

    END;

  UNTIL code = $1C;         {Abschluß mit RETURN}
END;


PROCEDURE Standardbild;
VAR i : byte;
BEGIN
   ClrScr;
   write(fullstern);
   GotoXY(0,2);
   write('xxxxxxxxxxxxxxxxxxxxxxxxxxxx M & T  SCHÜLERSPORT xxxxxxxxxxxxxxxxxxxxxxxxx');
   GotoXY(0,4);
   write('xxxxxxxxxxx (C)opyright 1990 by Markt & Technik Aktiengesellschaft xxxxxxxxxxx');
   GotoXY(0,6);
   write(fullstern);
   GotoXY(0,0);
   FOR i := 0 to 22 DO BEGIN
     write(randstern);
```

```
      GotoXY(79,i);
      write(randstern);
    END;
    write(fullstern);
END;

PROCEDURE Name.anlegen(zl: byte);
VAR s : string;
BEGIN
  zeile := zl;
  gotoxy(Xn + seite, z);
  Eingabe(s,Xn+seite,z,12,1,'');
  Nachname := s;
  gotoxy(Xv + seite, z);
  Eingabe(s,Xv+seite,z,10,1,'');
  Vorname := s;
END;


PROCEDURE Name.ausgabe;
BEGIN
  gotoxy(Xn + seite, z);
  write(Nachname);
  gotoxy(Xv + seite, z);
  write(Vorname);
END;


{xxxxxxxxx Methoden des Objekttyps "Person"xxxxxxxx}

CONSTRUCTOR Person.init(nachn, vorn: string; gen: boolean);
BEGIN
  genus := gen;
  Xg    := tab[4];
  Name.init(nachn, vorn);
END;


FUNCTION Person.getgen: boolean;
BEGIN
  getgen := genus;
END;


PROCEDURE Person.anlegen(zl: byte);
VAR ch : char;
s : string;
BEGIN
  zeile := zl;
  Name.anlegen(zeile);
  gotoxy(Xg + seite, z);
    REPEAT                                          { genus einlesen }
      Eingabe(s,Xg+seite,z,1,1,'');
      ch := Upcase(s[1]);
```

```
    UNTIL (ch = 'M') or (ch = 'J');
    gotoxy(Xg + seite, z);
    IF ch = 'M' THEN
    BEGIN
      genus := TRUE;
      write('M');
    END
    ELSE
    BEGIN
      genus := FALSE;
      write(' J');
    END;
END;


PROCEDURE Person.ausgabe;
BEGIN
  gotoxy(Xg + seite, z);
     IF genus = TRUE THEN write ('M')
       ELSE write(' J');
  Name.ausgabe;
END;


{xxxxxxxxx Methoden von "Teilnehmer" xxxxxxxxxxxxxxxxx}

CONSTRUCTOR Teilnehmer.init(nachn, vorn: string; gen: boolean;
                       ang, tlg: boolean; lz: real);
BEGIN
  angemeldet   := ang;
  teilgenommen := tlg;
  laufzeit     := lz;
  Xa           := tab[5];
  Xt           := tab[6];
  Xl           := tab[7];
  Person.init(nachn, vorn, gen);
END;


FUNCTION Teilnehmer.getang:boolean;
BEGIN
  getang := angemeldet;
END;


FUNCTION Teilnehmer.gettlg: boolean;
BEGIN
  gettlg := teilgenommen;
END;


FUNCTION Teilnehmer.getlz: real;
BEGIN
  getlz := laufzeit;
```

```
END;


PROCEDURE Teilnehmer.tg;
BEGIN
  teilgenommen := TRUE;
END;


PROCEDURE Teilnehmer.ntg;
BEGIN
  teilgenommen := FALSE;
END;


PROCEDURE Teilnehmer.ummelden;
BEGIN
  angemeldet := NOT angemeldet;
  IF angemeldet = FALSE THEN laufzeit := 0.00;
END;


PROCEDURE Teilnehmer.lieszeit;
VAR
 t : real;
 s : string;
 i : integer;
BEGIN
  IF angemeldet = FALSE THEN laufzeit := 0.00   ELSE
  BEGIN
    IF teilgenommen = FALSE THEN laufzeit := zeit_8   ELSE
    BEGIN
      gotoxy(Xl + seite, z);
      readln(s);
      val(s, t, i);
      IF (t >= 3) and (t < 10.00) and (i = 0) and (frac(t) < 0.6) THEN BEGIN
        laufzeit := t
      END
      ELSE BEGIN
        gotoxy(1, 23);
        write('xxxxxxxxxxxxxxxxxxxxxxxxxxxx Falsche Eingabe!!! xxxxxxxxxxxxxxxxxxxxxxxxxxxx');
        lieszeit;
      END;
    END;
  END;
END;


PROCEDURE Teilnehmer.ausgabe;
BEGIN
  Person.ausgabe;
  gotoxy(Xa + seite, z); IF angemeldet = TRUE THEN write('J')
                                                ELSE write('N');
  gotoxy(Xt + seite, z); IF teilgenommen = TRUE THEN write('J')
```

```
                                                   ELSE write('N');
  gotoxy(Xl + seite, z); write(laufzeit:4:2);
END;


{xxxxxxxxxxxxxxxxxx Methoden von "Schueler" xxxxxxxxxxxxxxxxxxxxxxxxxx}

CONSTRUCTOR Schueler.init(nachn, vorn, klbz: string; bon, ang, tlg: boolean;
                          lz: real; nr, klnr: byte);

BEGIN
    Klassenbezeichnung  := klbz;
    Xkbz                := 30;
    Zkbz                := 1;
    nummer              := nr;
    Klassennummer       := klnr;
    Xnr                 := tab[1];
    Teilnehmer.init(nachn, vorn, bon, ang, tlg, lz);
END;


FUNCTION Schueler.getklbz: string;
BEGIN
  getklbz := Klassenbezeichnung;
END;


FUNCTION Schueler.getnr: byte;
BEGIN
  getnr := nummer;
END;


FUNCTION Schueler.getklnr: byte;
BEGIN
  getklnr := Klassennummer;
END;


DESTRUCTOR Schueler.done;
BEGIN
END;


PROCEDURE Schueler.ausgabe;
BEGIN
  gotoxy(Xnr + seite, z);
  write(nummer);
  Teilnehmer.ausgabe;
END;


PROCEDURE Schueler.ausgabe2;
BEGIN
```

```
  gotoxy(Xnr + seite, z);
  write(nummer);
  Name.ausgabe;
  gotoxy(Xg + seite, z);
  write(Klassenbezeichnung);
  gotoxy(Xl + seite, z);
  write(laufzeit:4:2);
END;


PROCEDURE Schueler.anlegen(zl: byte);
VAR

  s    : string;
  nr   : longint;
  code : integer;
BEGIN
  zeile := zl;
  REPEAT
    gotoxy(Xnr + seite, z);
    Eingabe(s,xnr+seite,z,3,0,'');
    val(s, nr, code);
  UNTIL (code = 0) and (nr >= 0) and (nr <= 255);
  nummer := nr;
  Person.anlegen(zeile);
END;


PROCEDURE Schueler.maske;
CONST
  text: array[1..7] of string[20] =
      ('Nr ', 'Nachname    ','Vorname  ','M/J  ','A ', 'T ', 'Zeit  ');

VAR
  i: byte;
BEGIN
  ClrScr;
  gotoxy(1,2);
  FOR i := 1 to 7 DO
  BEGIN
    write(text[i]);
  END;
  FOR i := 1 to 7 DO
  BEGIN
    write(text[i]);
  END;
END;


{xxxxxxxxxxxxxxxxxx Methoden von Jahrgangsliste xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}

CONSTRUCTOR Jahrgangsliste.init;
BEGIN
END;
```

```
PROCEDURE Jahrgangsliste.add(VAR K: knPtr; S: scPtr);
VAR
  KN : knPtr;

BEGIN
    KN := NEW(knPtr);
    KN^.next := K;
    KN^.Item := S;
    K := KN;
END;


PROCEDURE Jahrgangsliste.sortieren(VAR K: knPtr);
VAR
  KN, L: knPtr;
  temp : scPtr;
BEGIN
  KN := K;
  WHILE KN^.next <> NIL DO
  BEGIN
    L := KN^.next;
    WHILE L <> NIL DO
    BEGIN
      IF KN^.item^.getnachn > L^.item^.getnachn THEN
      BEGIN
        temp := KN^.item;
        KN^.item := L^.item;
        L^.item := temp;
      END;
      L := L^.next;
    END;
  KN := KN^.next;
  END;
END;


PROCEDURE Jahrgangsliste.loeschen(klnr, nr: byte);
VAR
  K : knPtr;
BEGIN
  IF (root^.item^.getklnr = klnr) and (root^.item^.getnr = nr) THEN
  root := root^.next ELSE
  BEGIN
    K := root;
    WHILE K^.next <> NIL DO
    BEGIN
      IF (K^.next^.item^.getklnr = klnr) and (K^.next^.item^.getnr = nr) THEN
      K^.next := K^.next^.next;
      IF K^.next <> NIL THEN K := K^.next;
    END;
  END;
END;
```

```
PROCEDURE Jahrgangsliste.meldum(klnr, nr: byte);
VAR
  K : knPtr;
BEGIN
  K := root;
  WHILE K <> NIL DO
  BEGIN
    IF (K^.item^.getklnr = klnr) and (K^.item^.getnr = nr) THEN
      BEGIN
        K^.item^.ummelden;
        exit;
      END
    ELSE
      K := K^.next;
  END;
END;


PROCEDURE Jahrgangsliste.ausgeben;
VAR
  K : knPtr;
  N : Teilnehmer;
  sc: Schueler;
  i : byte;
BEGIN
  K := root;
  i := anfangszeile;
  WHILE K <> NIL DO              { zuerst die richtigen Zeilen setzen }
  BEGIN
    K^.item^.putzeile(i);
    inc(i);
    K := K^.next;
  END;
  sc.maske;
  K := root;
  WHILE K <> NIL DO
  BEGIN
    K^.item^.ausgabe;
    K := K^.next;
    inc(i);
  END;
END;


PROCEDURE Jahrgangsliste.load;

VAR
  s : string;
  scdat : Schuelerdaten;
  f     : file of Schuelerdaten;
BEGIN
  GotoXY(15,22);
```

```
  write('Dateiname : <                                            >');
  Eingabe(s,28,22,50,1,'');
  assign(f,s);
  {$I-}                          { Fehlererkennung aus }
  reset(f);
  { $I+}                         {           ...    ein }
  IF ioResult = 0 THEN
  BEGIN
    WHILE not EOF(f) DO
    BEGIN
      read(f, scdat);
      with scdat DO
      BEGIN
        add(root,NEW(scPtr,init(nachname, vorname, Klassenbezeichnung,
        genus, angemeldet, teilgenommen, laufzeit, nummer, Klassennummer)));
      END;
    END;
    sortieren(root);
    close(f);
  END
  ELSE
  BEGIN
    GotoXY(15,22);
    write('Dateifehler - bitte <RTN> drücken!                          ');
    readln;
  END;
Standardbild;
END;


PROCEDURE Jahrgangsliste.store;
VAR
  s     : string;
  K     : knPtr;
  scdat : Schuelerdaten;
  f     : file of Schuelerdaten;
BEGIN
  GotoXY(15,22);
  write('Dateiname : <                                            >');
  Eingabe(s,28,22,50,1,'');
  assign(f,s);
  {$I-}                         { Fehlerkennung aus }
  rewrite(f);
  {$I+}                         {        ...wieder an}
  IF ioResult = 0 THEN
  BEGIN
    K := root;
    WHILE K <> NIL DO
    BEGIN
        with scdat DO
        BEGIN
          Klassenbezeichnung := K^.item^.getklbz;
          Klassennummer      := K^.item^.getklnr;
          nummer             := K^.item^.getnr;
```

```
        nachname         := K^.item^.getnachn;
        vorname          := K^.item^.getvorn;
        genus            := K^.item^.getgen;
        angemeldet       := K^.item^.getang;
        teilgenommen     := K^.item^.gettlg;
        laufzeit         := K^.item^.getlz;
      END;
      write(f, scdat);
      K := K^.next;
    END;
    close(f);
  END
  ELSE
  BEGIN
    GotoXY(15,22);
    write('Dateifehler - bitte <RTN> drücken!                    ');
    readln;
  END;
END;


DESTRUCTOR Jahrgangsliste.done;
VAR
  K : knPtr;
BEGIN
  WHILE root <> NIL DO
  BEGIN
    K := root;
    dispose(K^.item, done);
    root := K^.next;
    dispose(K);
  END;
END;


{xxxxxxxxxxxxxxxxxxxxxxx Methoden von Jahrgangsbestenliste xxxxxxxxxxxxxxxxxxxxxxxxxx}

CONSTRUCTOR Jahrgangsbestenliste.init;
BEGIN
  start := NIL;
  Jahrgangsliste.init;
END;


PROCEDURE Jahrgangsbestenliste.einsammeln(gen: boolean);
VAR
  A : knPtr;
  S : scPtr;
BEGIN
  A  := root;
  WHILE (A <> NIL) DO
  BEGIN
    IF (A^.Item^.getgen = gen) and (A^.Item^.getang = TRUE)
       and (A^.Item^.gettlg = TRUE) and (A^.Item^.getlz > 3)
```

```
      THEN                          { nur tatsächliche Teilnehmer zählen }
      BEGIN                         { neuen Zeiger anlegen }
        S := NEW(scPtr, init(A^.item^.getnachn, A^.item^.getvorn,
                 A^.item^.getklbz, A^.item^.getgen, A^.item^.getang,
                 A^.item^.gettlg, A^.item^.getlz, A^.item^.getnr,
                 A^.item^.getklnr));
        add(start, S);              { die Liste verlängern }
      END;
      A := A^.next;
  END;
END;


PROCEDURE Jahrgangsbestenliste.sortieren(VAR K: knPtr);
VAR
  KN, L: knPtr;
  temp : scPtr;
BEGIN
  KN := K;
  WHILE KN^.next <> NIL DO
  BEGIN
    L := KN^.next;
    WHILE L <> NIL DO
    BEGIN
      IF KN^.item^.getlz > L^.item^.getlz THEN
      BEGIN
        temp := KN^.item;
        KN^.item := L^.item;
        L^.item := temp;
      END;
      L := L^.next;
    END;
  KN := KN^.next;
  END;
END;


PROCEDURE Jahrgangsbestenliste.ausgeben;
VAR
  S : knPtr;
  i : integer;
BEGIN                             { zuerst richtige Platzziffern festlegen }
  sortieren(start);
  S := start;
  i := 1;
  WHILE S <> NIL DO
  BEGIN
    S^.item^.nummer := i;
    IF (S^.item^.getlz <> S^.next^.item^.getlz) THEN
      inc(i);                     { gleiche Zeiten - gleiche Ziffern }
    S := S^.next;
  END;

  S := start;
```

```
  i := anfangszeile;
  WHILE S <> NIL DO          { zuerst die richtigen Zeilen festlegen }
  BEGIN
    IF S^.item^.getgen = FALSE THEN
      S^.Item^.putzeile(i + maxzeile - anfangszeile)
    ELSE
      S^.Item^.putzeile(i);
    S := S^.next;
    inc(i);
  END;
  gotoxy(15, 2);
  write('Mädchen              Bestenliste              Jungen');
  S := start;
  WHILE S <> NIL DO          { Ausgabe der Liste }
  BEGIN
    S^.item^.ausgabe2;
    S := S^.next;
  END;
END;


DESTRUCTOR Jahrgangsbestenliste.done;
VAR
  K : knPtr;
BEGIN
  WHILE start <> NIL DO
  BEGIN
    K := start;
    dispose(K^.item, done);
    start := K^.next;
    dispose(K);
  END;
END;


PROCEDURE Jahrgangsbestenliste.herstellen;  { eine Art Hauptprogramm }
BEGIN
  einsammeln(TRUE);
  ausgeben;
  done;
  einsammeln(FALSE);
  ausgeben;
  done;
END;


{xxxxxxxxxxxxxxxxxxxxx Methoden von Klassenbestenliste xxxxxxxxxxxxxxxxxxxxxx}

CONSTRUCTOR Klassenbestenliste.init;
BEGIN
  Jahrgangsbestenliste.init;
END;
```

```
PROCEDURE Klassenbestenliste.einsammeln(gen: boolean; klnr: byte);
VAR
  A : knPtr;
  S : scPtr;
BEGIN
  A  := root;
  WHILE (A <> NIL) DO
  BEGIN
    IF (A^.Item^.getgen = gen) and (A^.Item^.getang = TRUE)
       and (A^.Item^.gettlg = TRUE) and (A^.Item^.getlz > 3)
       and (A^.item^.getklnr = klnr)
       THEN                         { nur tatsächliche Teilnehmer zählen }
       BEGIN                        { neuen Zeiger anlegen }
         S := NEW(scPtr, init(A^.item^.getnachn, A^.item^.getvorn,
                          A^.item^.getklbz, A^.item^.getgen, A^.item^.getang,
                          A^.item^.gettlg, A^.item^.getlz, A^.item^.getnr,
                          A^.item^.getklnr));
         add(start, S);                { die Liste verlängern }
       END;
       A := A^.next;
  END;
END;

PROCEDURE Klassenbestenliste.herstellen(klnr: byte); {eine Art Hauptprogramm}
BEGIN
  einsammeln(TRUE, klnr);
  ausgeben;
  done;
  einsammeln(FALSE, klnr);
  ausgeben;
  done;
END;


{xxxxxxxxxxxxxxxxxxxxx Methoden von Klassenliste xxxxxxxxxxxxxxxxxxxxxxxxxxxxx}

CONSTRUCTOR Klassenliste.init;
BEGIN
  Jahrgangsliste.init;
END;


FUNCTION Klassenliste.ausgeben(klnr: byte): knPtr;
VAR
  S, temp : knPtr;
  i : integer;
  sc : Schueler;
BEGIN
  sortieren(root);          { die sortierte Liste bearbeiten }
  S := root;
  i := anfangszeile;
  WHILE S <> NIL DO         { zuerst die richtigen Zeilen festlegen }
  BEGIN
    IF S^.item^.getklnr = klnr THEN
```

```
      BEGIN
        S^.Item^.putzeile(i);
        inc(i);
        S := S^.next;
      END
    ELSE S := S^.next;
  END;

  sc.maske;
  S := root;
  temp := root;
  WHILE S <> NIL DO             { Liste ausgeben }
  BEGIN
    IF S^.item^.getklnr = klnr THEN
      BEGIN
        S^.Item^.ausgabe;
        temp := S;
        S := S^.next;
      END
    ELSE S := S^.next;
  END;
  ausgeben := temp;
  IF temp <> root THEN
  BEGIN
    gotoxy(20, 1);
    write('Klassenbezeichnung:    ', temp^.item^.getklbz);
  END;
END;


PROCEDURE Klassenliste.listeanlegen(klnr: byte);
VAR
  K     : knPtr;
  N     : scPtr;
  sc    : Schueler;
  klbz,s : string;
  ch    : char;
  zl    : byte;
  jgl   : Jahrgangsliste;
BEGIN
  sc.maske;
  K := ausgeben(klnr);          { zuerst mal zeigen, was da ist }
  IF K = root THEN
    BEGIN
      zl := anfangszeile;
      gotoxy(20, 1);
      write('Klassenbezeichnung :  <   >');
      Eingabe(klbz,43,1,3,1,'');
    END
  ELSE
    BEGIN
      zl := K^.item^.getzeile + 1; { bei der nächsten Zeile geht es weiter }
      klbz := K^.item^.getklbz;
    END;
```

```
  REPEAT
    N := NEW(scPtr, init('', '',klbz, TRUE, TRUE, FALSE, zeit_8, 1, klnr));
    N^.anlegen(zl);
    gotoxy(1, 23);
    write('xxxxxxxxxxxxxxxxxx Weiter: <W>   Abbruch: <A>   Menue: <M> xxxxxxxxxxxxxxxxxx');
    REPEAT
      ch := Upcase(chr(LesTaste(ASCII)));
    UNTIL (ch = 'W') or (ch = 'A') or (ch = 'M');
    gotoxy(1, 25);
    IF ch = 'A' THEN exit;                { zurück ohne zu speichern }
    jgl.add(root, N);
    IF ch = 'W' THEN inc(zl);             { weiter eingeben }
    IF ch = 'M' THEN                      { aufhören }
       BEGIN
         gotoxy(1, 23);
         write('xxxxxxxxxxxxxxxxxxxx Soll neu numeriert werden? (J/N) xxxxxxxxxxxxxxxxxxxx');
         REPEAT
           ch := upcase(chr(LesTaste(ASCII)));
         UNTIL (ch = 'J') or (ch = 'N');
         IF ch = 'J' THEN numerieren(klnr);
         exit;
       END;
  UNTIL zl = 50;
END;


PROCEDURE Klassenliste.numerieren(klnr: byte);
VAR
  i : byte;
  K : knPtr;
  jl: Jahrgangsliste;
BEGIN
  sortieren(root);
  K := root;
  i := 1;
  WHILE K <> NIL DO
  BEGIN
    IF K^.item^.getklnr = klnr THEN
       BEGIN
         K^.Item^.nummer:= i;
         inc(i);
         K := K^.next;
       END
    ELSE K := K^.next;
  END;
END;


PROCEDURE Klassenliste.loeschen(klnr: byte);
VAR
  K     : knPtr;
  jgl   : Jahrgangsliste;
  i     : byte;
```

```
  nr    : longint;
  s     : string;
  code  : integer;
  ch    : char;
BEGIN
  K := ausgeben(klnr);
  IF K = root THEN
  BEGIN
    gotoxy(5, 5);
    write ('Keine Klassenliste vorhanden!!  <RTN>  drücken!');
    readln;
    exit;
  END;
  gotoxy(1, 23);
  write('xxxxxxxxxxxxxx Welche Nummer soll gelöscht werden? : <   > xxxxxxxxxxxxxxx');
  REPEAT
    Eingabe(s,56,23,3,0,'');
    val(s, nr, code);
  UNTIL (code = 0) and (nr > 0) and (nr <= 255);
  jgl.loeschen (klnr, nr);
  K := ausgeben(klnr);
  gotoxy(1, 23);
  write('xxxxxxxxxxxxxxxxxxxxxxxxxx Weiter: <W>   Menue: <M> xxxxxxxxxxxxxxxxxxxxxxxxxx');
  REPEAT
    ch := upcase(chr(LesTaste(ASCII)));
  UNTIL (ch = 'W') or (ch = 'M');
  IF ch = 'M' THEN                                  { aufhören }
  BEGIN
    gotoxy(1, 23);
    write('xxxxxxxxxxxxxxxxxxxxxx Soll neu numeriert werden? (J/N) xxxxxxxxxxxxxxxxxxxxx');
    REPEAT
      ch := upcase(chr(LesTaste(ASCII)));
    UNTIL (ch = 'J') or (ch = 'N');
    IF ch = 'J' THEN numerieren(klnr);
    exit;
  END;
  IF ch =  'W' THEN loeschen(klnr);
END;


PROCEDURE Klassenliste.ummelden(klnr: byte);
VAR
  K     : knPtr;
  i     : byte;
  nr    : longint;
  ch    : char;
  s     : string;
  code  : integer;
BEGIN
  K := ausgeben(klnr);
  IF K = root THEN
  BEGIN
    gotoxy(5, 5);
    write ('Keine Klassenliste vorhanden!!  <RTN>  drücken!');
```

```
    readln;
    exit;
  END;
  gotoxy(1, 23);
  write('xxxxxxxxxxxxx Welche Nummer soll umgemeldet werden? : <   > xxxxxxxxxxxxxxxxx');
    REPEAT
      Eingabe(s,57,23,3,0,'');
      val(s, nr, code);
    UNTIL (code = 0) and (nr > 0) and (nr <= 255);
    meldum(klnr, nr);
    K := ausgeben(klnr);
    gotoxy(1, 23);
    write('xxxxxxxxxxxxxxxxxxxxxxxxxx Weiter: <W>   Menue: <M> xxxxxxxxxxxxxxxxxxxxxxxxxx');
    REPEAT
      ch := upcase(chr(LesTaste(ASCII)));
    UNTIL (ch = 'W') or (ch = 'M');
    IF ch = 'M' THEN exit;
    IF ch = 'W' THEN ummelden(klnr);
END;


PROCEDURE Klassenliste.zeiteingeben(klnr: byte);
VAR
  S    : scPtr;
  ch   : char;
  K    : knPtr;
  ss   : string;

FUNCTION springen: knPtr;
VAR
  s    : string;
  nr   : byte;
  K    : knPtr;
  code : word;
BEGIN
  gotoxy(1, 23);
  write('xxxxxxxxxx Zu welcher Nummer soll gesprungen werden? : <   > xxxxxxxxxx');
  Eingabe(s,58,23,3,0,'');
  val(s,nr,code);
  K := root;
  WHILE K <> NIL DO
  BEGIN
    IF (K^.item^.getklnr = klnr) and (K^.item^.getnr = nr) THEN
      BEGIN
        springen := K;
        exit;
      END
    ELSE  K := K^.next;
  END;
  springen := K;
END;
```

```
BEGIN
  K := ausgeben(klnr);           { zuerst die ganze Liste ausgeben }
  K := root;
  WHILE K <> NIL  DO
  BEGIN
    IF (K^.item^.getklnr = klnr) THEN
    BEGIN
      IF (K^.item^.getang = TRUE) THEN
      BEGIN
        gotoxy(1, 23);
        write('xxxxxxxxxxxxxx Springen: <S> Weiter: <W> Teilgenommen: <J/N> xxxxxxxxxxxx');
        gotoxy(K^.item^.Xt + K^.item^.seite, K^.item^.z);   {an die r. Pos.}
        REPEAT
          ch := Upcase(chr(LesTaste(ASCII)));
        UNTIL (ch = 'S') or (ch = 'W') or (ch = 'J') or (ch = 'N');
        IF ch = 'S' THEN
        BEGIN
          K := springen;
          IF K = NIL THEN exit;
          gotoxy(1, 23);
          write('xxxxxxxxxxxxxxxxxx Weiter: <W>   Teilgenommen: <J/N>  xxxxxxxxxxxxxxxxxx');
          gotoxy(K^.item^.Xt + K^.item^.seite, K^.item^.z); {an die r. Pos.}
          REPEAT
            ch := Upcase(chr(LesTaste(ASCII)));
          UNTIL (ch = 'W') or (ch = 'J') or (ch = 'N');
          IF ch = 'J' THEN K^.item^.tg;
          IF ch = 'N' THEN K^.item^.ntg;
          K^.item^.ausgabe; K^.item^.lieszeit; K^.item^.ausgabe;
        END
        ELSE
        BEGIN
          IF ch = 'J' THEN K^.item^.tg;
          IF ch = 'N' THEN K^.item^.ntg;
          K^.item^.ausgabe; K^.item^.lieszeit; K^.item^.ausgabe;
        END;
        gotoxy (1, 23);
        write('xxxxxxxxxxxxxxxxxxxxxxxxx Weiter: <W>  Menü: <M>  xxxxxxxxxxxxxxxxxxxxxxxxx');
        REPEAT
          ch := upcase(chr(LesTaste(ASCII)));
        UNTIL (ch = 'W') or (ch = 'M');
        IF ch = 'M' THEN exit;
      END
      ELSE K^.item^.lieszeit;
    END;
    K := K^.next;
  END;
END;
```

```
FUNCTION Klassenliste.durchschnitt(klnr: byte): string;
VAR
  K : knPtr;
  i : byte;                      { Zählvariable }
  sum : real;                    { Summe }
  text1, text2 : string[2];
FUNCTION sekunde: real;          { wandelt die eingetragene Laufzeit..}
BEGIN                             { .. in Sekunden um }
  sekunde := trunc(K^.item^.getlz) * 60 + frac(K^.item^.getlz) * 100;
END;
BEGIN
  K := root; i := 0; sum := 0;    { Initialisierungen }
  WHILE K <> NIL DO
  BEGIN
    IF (K^.item^.getang = TRUE) and (K^.item^.getklnr = klnr) THEN
      BEGIN                       { nur angemeldete Teilnehmer zählen }
      sum := sum + sekunde;
      IF K^.item^.getgen = TRUE THEN  (Mädchen bekommen einen Bonus }
      sum := sum - bon_8;
      inc(i);
    END;
    K := K^.next;
  END;
  IF i = 0 THEN i := 1;           { im Fall des Falles }
  sum := (sum/i)/60;              { in Minuten umwandeln }
  str(trunc(sum), text1);
  str(frac(sum)*60:2:0, text2);
  durchschnitt:= concat(text1, ' min ', text2, ' sec'); {als Str. ausgeben}
END;

{xxxxxxxxxxxxxxxxEnde der Methoden xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}


FUNCTION probe: byte;
VAR
  s    : string;
  klnr : longint;
  code : integer;
BEGIN
  REPEAT
    gotoxy(15, 22);
    write('Bitte die Klassennummer eingeben : <   >');
    Eingabe(s,51,22,3,0,'');
    val(s, klnr, code);
  UNTIL (code = 0) and (klnr > 0) and (klnr <= 255);
  probe := klnr;
END;


PROCEDURE Menue;
CONST
  sp = 15;
VAR
  klnr : byte;
```

```
BEGIN
  gotoxy(sp, 8);
  write('Bitte wählen Sie :');
  gotoxy(sp, 10);
  write('<1> Datei laden');
  gotoxy(sp,11);
  write('<2> Datei speichern');
  gotoxy(sp,12);
  write('<3> Klassendaten eingeben   ');
  gotoxy(sp, 13);
  write('<4> Klassendaten ausgeben  ');
  gotoxy(sp, 14);
  write('<5> Daten löschen    ');
  gotoxy(sp, 15);
  write('<6> Ummeldungen    ');
  gotoxy(sp, 16);
  write('<7> Zeit eingeben      ');
  gotoxy(sp, 17);
  write('<8> Jahrgangsbesten ');
  gotoxy(sp, 18);
  write('<9> Klassenbesten  ');
  gotoxy(sp, 19);
  write('<0> ENDE   ');
  gotoxy(sp, 20);
  write('Ihre Wahl : < >');
  REPEAT
    Eingabe(wahl,sp+13,20,1,0,'');
  UNTIL wahl[1] in ['1', '2', '3', '4', '5', '6', '7', '8','9','0'];
  case wahl[1] of
  '1': BEGIN
        root := NIL;
        Einejahrgangsliste.init;
        Einejahrgangsliste.load;
       END;
  '2': IF root<>NIL THEN BEGIN
        Einejahrgangsliste.store;
       END;
  '3': BEGIN
         klnr := probe;
         Eineklassenliste.init;
         Eineklassenliste.listeanlegen(klnr);
       END;
  '4': IF root <> NIL THEN BEGIN
         klnr := probe;
         Eineklassenliste.init;
         SCR := Eineklassenliste.ausgeben(klnr);
         gotoxy(1, 23);
         write('xxxxxxx Klassendurchschnitt :'
                ,Eineklassenliste.durchschnitt(klnr));
         gotoxy(48, 23);
         write('Bitte <RTN> eingeben xxxxxxx');
         readln;
       END;
   '5': IF root<>NIL THEN BEGIN
```

```
        klnr := probe;
        Eineklassenliste.init;
        Eineklassenliste.loeschen(klnr);
       END;
  '6': IF root<>NIL THEN BEGIN
        klnr := probe;
        Eineklassenliste.init;
        Eineklassenliste.ummelden(klnr);
       END;
  '7': IF root<>NIL THEN BEGIN
        klnr := probe;
        Eineklassenliste.init;
        Eineklassenliste.zeiteingeben(klnr);
       END;
  '8': IF root<>NIL THEN BEGIN
        ClrScr;
        Einejahrgangsbestenliste.init;
        Einejahrgangsbestenliste.herstellen;
        gotoxy(1, 23);
        write('xxxxxxxxxxxxxxxxxxxxxxxxxxx Bitte <RTN>',
               'eingeben xxxxxxxxxxxxxxxxxxxxxxxxxxx');
        readln;
       END;
  '9': IF root<> NIL THEN BEGIN
        klnr := probe;
        ClrScr;
        Eineklassenbestenliste.init;
        Eineklassenbestenliste.herstellen(klnr);
        gotoxy(1, 23);
        write('xxxxxxxxxxxxxxxxxxxxxxxxxxx Bitte <RTN>',
               'eingeben xxxxxxxxxxxxxxxxxxxxxxxxxxx');
        readln;
       END;
  END;
END;


{xxxxxxxxHauptprogrammxxxxxxxxx}

BEGIN
Textmode(dynamisch);
root := NIL;
REPEAT
  ClrScr;
  Standardbild;
  menue;
UNTIL wahl = '0';
ClrScr;
END.
```

Natürlich kann ich an dieser Stelle nicht die OOP-Theorie erläutern, da der Rahmen des Buches damit gesprengt würde. Ich möchte Ihnen jedoch dringend empfehlen, an Hand der Original-Anleitung zu Turbo Pascal 5.5 das Programm nachzuvollziehen, da diese Anleitung für mich das Beste zum Thema OOP ist. Zum einen haben die Autoren verstanden, wovon sie reden (sonst hätten sie ja auch kaum den Compiler entwickeln können), zum anderen ist das Werk relativ kurzgehalten, so daß man es in ein paar Tagen durchlesen kann.

Nun jedoch zum Programm SCHUELER.EXE. Zur Abwechslung arbeitet es im dynamischen Bildschirmmodus mit 80x25 Zeichen. Wie schon oben erwähnt, dient es zur Erfassung von Sportwettkämpfen in Schulen, wie z.B. den Bundesjugendspielen. Betrachtet werden einzelne Jahrgänge (alle Schüler, die in der gleichen Klassenstufe sind) und die einzelnen Klassen. Wenn es z.B. die Klassen 8a, 8b und 8c mit jeweils 20 Schülern gibt, würde der Jahrgang aus drei Klassen und 60 Schülern bestehen.

Über den Menüpunkt [3] können Schülerdaten einer Klasse eingegeben werden. Dazu werden Sie zunächst nach der Klassennummer befragt. Diese hat mit der tatsächlichen Bezeichnung nichts zu tun, sondern ist eine rein logische Nummer, mit der die Klassen durchnumeriert werden. Die erste Klasse sollte man daher mit »1«, die zweite mit »2« usw. bezeichnen. Die einzelnen Schüler bestehen aus einer Nummer, dem Vor- und Nachnamen sowie einem Geschlecht, das entweder mit »m« für Mädchen oder mit »j« für Junge angegeben werden kann. Nach der Eingabe können Sie weitere Schüler eingeben ([W]), zurück zum Menü springen ([M]) oder abbrechen ([A]). Vor dem Rücksprung ins Menü kann wahlweise noch neu durchnumeriert werden, die Schüler werden alphabetisch sortiert.

Über den Menüpunkt [5] können Schüler gelöscht werden, indem ihre Nummer eingegeben wird. Interessant wird es mit dem Menüpunkt [7]: Hier können für die einzelnen Schüler die erreichten Laufzeiten eingegeben werden. Während unter der Tabellenspalte »T« (Teilgenommen) ein »J« für »Ja« erscheint, steht unter der Spalte »A« (Angekommen) ein »N« für »Nein« und als Zeit der indiskutabel schlechte Wert von 8 Minuten und 30 Sekunden. Sie können nun das »N« in der Angekommen-Spalte in ein »J« verwandeln und die Laufzeit in die Zeit-Spalte eintragen, wobei unerreichbar gute oder schlechte Zeiten nicht zugelassen werden.

Nach diesen Eingaben können Sie sich über die Menüpunkte [8] und [9] die Bestenliste der Klasse bzw. des Jahrgangs, d.h. aller Klassen, getrennt nach Jungen und Mädchen ansehen. Sicherlich werden Sie einwenden, daß dieses Programm nicht gerade sinnvoll in der Praxis einzusetzen ist,

es sein denn, für Lehrer. Es stellt aber eine Anwendung dar, die für die OOP-Programmierung wie geschaffen ist, da man mit dem Objekt »Schüler« sinnvoll arbeiten kann. Zudem wird die Erstellung einer Baumstruktur verdeutlicht, wie sie in der Informatik eine große Rolle spielt (Dynamische Datenverwaltung).

Kapitel 3: Programmieren in C

Natürlich ist es auch möglich, den Portfolio in Turbo C zu programmieren. Mit Turbo C ist in diesem Fall die Version 2.0 des Compilers gemeint. Da auch der C-Compiler einen Umfang besitzt, der es unmöglich macht, ihn auf dem Portfolio selbst zum Laufen zu bringen, muß das Programm auf dem PC geschrieben und übersetzt werden. Auf die Unterschiede der Arbeitsweise der C-Funktionen auf dem PC und dem Portfolio und der Programmierung einer »Portfolio-üblichen« Benutzeroberfläche soll in diesem Kapitel anhand eines etwas größeren Beispielprogramms eingegangen werden.

3.1 Vorbemerkungen

3.1.1 Notation

C erlaubt dem Programmierer große Freiheiten bei der Schreibweise seiner Programme. Wenn man will, kann ein C-Listing aussehen, als ob jemand »ein Gürteltier über die Tastatur gewälzt hätte«. Speziell in der Anfangszeit war C eine Sprache der Unix-Gurus, die sich nicht gern in die Karten sehen lassen wollten und deswegen ihre Programme sehr kryptisch geschrieben hatten. Nun ist dieses aber für ein Buch wie das vorliegende wenig hilfreich. Damit Sie die Listings auch verstehen können, wurde weitestgehend die Notation von Kernighan und Ritchie, den »C-Erfindern«, verwendet. Aus Platzgründen mußten aber öfter mehrere Befehle auf eine Zeile geschrieben werden. Wir hoffen aber, daß die umfangreichen Kommentarzeilen Ihnen helfen, die Vorgehensweise zu verstehen.

Ungewöhnlich kann Ihnen die Bezeichnung der Variablen vorkommen. Es steckt aber ein Sinn dahinter, der gleich erläutert wird. Die Variablenbezeichnung beginnt grundsätzlich mit einem Großbuchstaben oder einer Zahl. Vor dieser Bezeichnung steht ein Präfix aus Kleinbuchstaben, die den Typ der Variablen spezifizieren. In der folgenden Tabelle sind alle Möglichkeiten aufgeführt.

Buchstabe	Bedeutung	Erklärung
a	array	bezeichnet ein Feld des folgenden Variablentyps
b	BYTE	vorzeichenlose Variable mit 1 Byte Länge (unsigned char)
c	CHAR	vorzeichenbehaftete Variable mit 1 Byte Länge
f	FLOAT	vorzeichenbehaftete Fließkommavariable
i	INT	vorzeichenbehaftete Variable mit 2 Byte Länge (short)
l	LONG	vorzeichenbehaftete Variable mit 4 Byte Länge
p	pointer	Zeiger auf den folgenden Variablentyp

Finden Sie also im Quelltext eine Variable mit Namen *aplPointerArray*, so bezeichnet diese ein Feld mit Zeigern auf LONG-Variablen. Deklariert wurde diese Variable beispielsweise mit

```
LONG     *aplPointerArray[20];
```

Die beliebten Zeiger auf CHAR-Variablen werden dann beispielsweise als

```
CHAR     *pcPointer;
```

und INT-Variablen als

```
INT iTest;
```

deklariert.

Die Variablentypen wie *BYTE*, *LONG* oder *INT* werden immer großgeschrieben. Der Grund ist eine möglichst maschinenunabhängige Programmierung. Da speziell der Variablentyp *INT* eine maschinenabhängige Länge besitzt, wird im Beispielprogramm die Datei TYPEDEF.H eingebunden, in der mittels *#define*-Anweisungen neue Variablentypen definiert werden. Um ein Programm von einer zur anderen Maschine zu portieren, braucht in bezug auf die Variablen nichts im Programm geändert werden. Anpassungen müssen nur innerhalb der Datei TYPEDEF.H vorgenommen werden.

3.1.2 Compiler-Einstellungen

Bei den Compiler-Einstellungen gibt es nur wenig zu beachten. Es finden sich nur im Optionen-Menü einige Punkte, die speziell für die Erzeugung von Portfolio-Programmen gesetzt werden müssen.

Compiler/Code generation
Hier darf der Umschalter nur auf 8088/86 stehen. Da der Portfolio nur einen 80C88-Prozessor enthält, würde sonst teilweise Code erzeugt werden, der nicht ausgeführt werden kann.

Compiler/Floating point
Da es bisher keine Möglichkeit gibt, den Portfolio mit einem numerischen Koprozessor aufzurüsten, sind bei diesem Punkt nur *Emulation* oder *None* sinnvoll.

Compiler/Alignment
Beide Einstellungen liefern fehlerfreie Programme, doch BYTE bringt Ihnen einen kürzeren Programmcode, da die Variablen nicht auf gerade Adressen ausgerichtet werden, was den ohnehin knappen Speicherplatz des Portfolio nicht mehr als nötig verschwendet.

Compiler/Merge duplicate strings
Auch mit dieser Option läßt sich Speicherplatz sparen, in dem mehrfach vorkommende Strings nur einmal abgespeichert werden.

Compiler/Standard stack frame
Compiler/Test stack overflow
Compiler/Line numbers
Compiler/OBJ Debug information
Alle diese Optionen verlängern ein Programm, sind aber sehr hilfreich beim Debuggen. Schalten Sie sie also erst ab, wenn sich keine Fehler mehr im Programm befinden.

Linker/Graphics library
Es ist überflüssig, den Linker auch innerhalb der Grafikbibliothek nach Funktionen suchen zu lassen, da die Grafik des Portfolio nicht von Turbo C unterstützt wird.

3.2 Das Beispielprogramm DISVER.EXE

Für die Demonstration der Programmierung des Portfolio dient im weiteren das Beispielprogramm DISVER.EXE. Es stellt ein Hilfsmittel zur Verwaltung der Dateien auf den einzelnen RAM-Karten dar und schafft einen Überblick, welche Dateien auf welcher RAM-Karte zu finden sind.

3.2.1 Die Bedienung des Beispielprogramms

Nach dem Starten meldet sich das Programm mit dem Dateibildschirm, in dem später die gespeicherten Dateien angezeigt werden. Vorab ist gleich das Informationsfenster geöffnet, welches einiges über das Programm verrät. Durch Drücken einer Taste schließt das Fenster wieder und Sie können mit dem Programm arbeiten.



Bild 3.1 Beginn des Programms

Dateibildschirm
In ihm werden die gespeicherten Dateien angezeigt. Durch die Länge des Portfolio-Bildschirms ist die Anzeige auf acht Dateieinträge gleichzeitig begrenzt. Von jeder Datei werden der Name der Datei mit längstens zwölf Zeichen, die Länge in Byte mit maximal sechs Ziffern und der Name der Karte, auf der sich die Datei befindet, mit bis zu zehn Zeichen. Mit den Cursor-Tasten kann die Bildschirmanzeige um eine Datei nach oben bzw. unten gerollt werden. Um acht Dateien, also einen ganzen Bildschirm, können Sie mit den Tasten [Bild↑] und [Bild↓] blättern. Die Tasten [Pos1] bzw. [Ende] bringen Sie an den Anfang bzw. das Ende der Dateiliste. Natürlich haben diese Tasten nur eine Funktion, wenn mehr als acht Dateien gespeichert sind.

Dateimenü

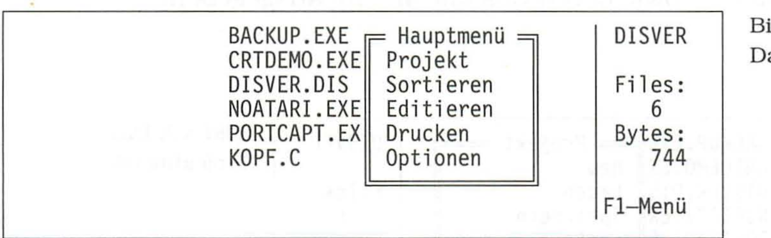


Bild 3.2: Das Dateimenü

Die verschiedenen Funktionen des Programms erreichen Sie über das Dateimenü, das »Portfolio-üblich« über die Funktionstaste [F1] aufgerufen wird. Es enthält die Menüpunkte:

- *Projekt*
- *Sortieren*
- *Editieren*
- *Drucken*
- *Optionen*

Auch die Auswahl der Menüpunkte geschieht so, wie Sie es bei den eingebauten Programmen gewohnt sind. Mit [Return] wird derjenige Punkt ausgewählt, auf dem der Blockcursor steht. Der Cursor kann mit Hilfe der Tasten [↑] und [↓] in der Menüliste auf und ab bewegt werden. Alternativ kann durch Eingabe des Anfangsbuchstabens ein Menüpunkt direkt angewählt werden.

Die Menüpunkte *Projekt*, *Sortieren*, *Editieren* und *Optionen* führen zu weiteren Untermenüs, die im folgenden erklärt sind. Nur der Punkt *DRUCKEN* hat eine direkte Aktion zur Folge.

Drucken

Es wird die komplette Dateiliste zweispaltig ausgedruckt, dabei sind die Dateien in der gleichen Reihenfolge sortiert wie auf dem Bildschirm. Voraussetzung ist natürlich eine parallele Schnittstelle mit einem angeschlossenen Drucker. Um das Programm nicht unnötig zu verlängern, wird nicht überprüft, ob ein Drucker angeschlossen und auch druckbereit ist. Falls Ihnen das nicht gefällt, können Sie leicht über den Aufruf des Interrupt 17hex direkt mit dem Drucker kommunizieren und seinen Status abfragen. An den Drucker selbst werden keine besonderen Anforderungen gestellt, er muß nur pro Zeile 80 Zeichen drucken können. Das Programm benutzt auch nur den Steuercode für diese Einstellung, so daß eventuelle Schriftarteinstellungen Ihrerseits weiterhin gültig bleiben. Als Kopfzeile

werden Druckdatum und aktuelle Seite, als Fußzeile die Anzahl der gespeicherten Dateien sowie deren Gesamtumfang ausgegeben.

Projektmenü

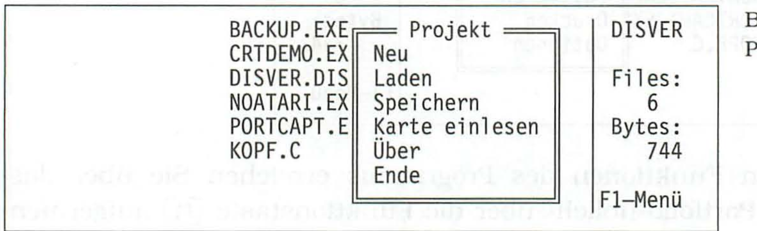


Bild 3.3: Das Projektmenü

Dieses besteht aus den Menüpunkten:

- *Neu*
- *Laden*
- *Speichern*
- *Karte einlesen*
- *Über*
- *Ende*

Der Punkt *Neu* dient zum Löschen aller im Speicher befindlichen Dateien. Mit *Laden* werden bereits vorher eingelesene und abgespeicherte Dateien wieder geladen und an das Ende der bereits im Speicher befindlichen Dateiliste angehängt. Zur Auswahl des Dateinamens erscheint ein neues Fenster, in dem der Dateiname und der Pfad zur Datei editiert werden kann.

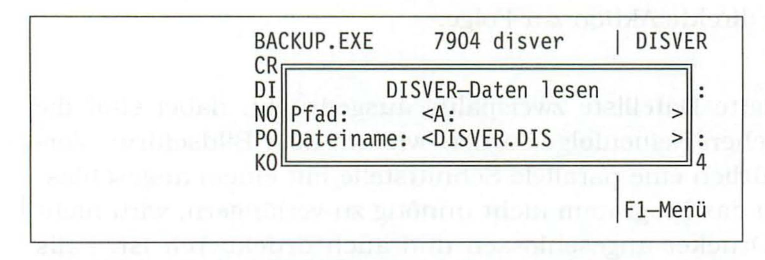


Bild 3.4: Abfrage des Dateinamens beim Laden

Speichern ist das Gegenstück zum vorherigen Menüpunkt. Hier kann die Dateiliste gespeichert werden. *Karte einlesen* springt zum Directory-Bildschirm, der den kompletten Verzeichnis-Baum des eingestellten Laufwerks einliest. Informationen über das Programm zeigt der Punkt *Über*. Er öffnet das Informationsfenster, das dem Benutzer mitteilt, welche Version des Programms vorliegt, wer das Programm erstellt hat, wie viele Dateien im

Speicher sind, wie lang alle Dateien zusammen sind und wie viele Dateien noch in die Dateiliste aufgenommen werden können. Durch Drücken einer Taste wird das Fenster wieder geschlossen. Da jedes Programm ein Ende haben muß, finden Sie es im Menüpunkt *Ende*. Sollte die Dateiliste vorher noch nicht gespeichert sein, werden Sie dazu noch aufgefordert.

Sortiermenü

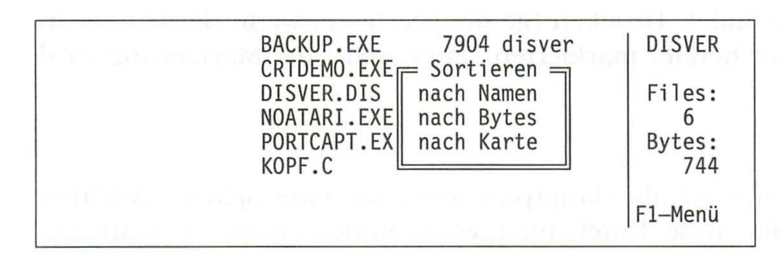


Bild 3.5: Das Sortiermenü

Je nach gewähltem Menüpunkt

- *nach Namen*
- *nach Bytes*
- *nach Karte*

wird die Dateiliste nach Dateinamen, der Länge in Byte oder dem Namen der Karte sortiert. Bedingt durch den verwendeten Algorithmus *Heapsort* kann der Vorgang einige Zeit in Anspruch nehmen.

Editiermenü



Bild 3.6: Das Editiermenü

Für das Ändern der Einträge in der Dateiliste stehen folgende Menüpunkte zur Verfügung:

- *Hauptprogramm*
- *Unterprogramm*
- *Editieren*

– Neues File
– Löschen

Für fast alle Funktionen dieses Menüs ist es notwendig, eine oder mehrere Dateien markiert zu haben. Sie bewegen den Cursor auf die gewünschte Datei und drücken die Leertaste. Daraufhin ist diese Datei markiert, als optische Kennzeichnung erscheint links neben dem Dateinamen ein nach rechts gerichtetes Dreieck (►, ASCII-Code 16) in der Spalte, in der sich auch der Cursor befindet. Drücken Sie die Leertaste, wenn der Cursor in einer Zeile mit einer bereits markierten Datei steht: die Markierung wird wieder gelöscht.

Hauptprogramm
Die markierte Datei wird als Hauptprogramm für eine spätere Addition markiert. Damit Sie diese Datei auch wiederfinden, wird sie statt mit dem Dreieck mit dem französischen Anführungsstrich gekennzeichnet (», ASCII-Code 175). Sollten versehentlich mehrere Dateien markiert sein, wird nur die erste Datei markiert, und alle anderen Markierungen werden gelöscht.

Unterprogramm
Hiermit wird die Dateilänge, einer oder mehrerer markierter Dateien, zu der Dateilänge einer vorher als Hauptprogramm markierten Datei addiert. Die Unterprogrammdateien werden im Anschluß gelöscht.

Editieren

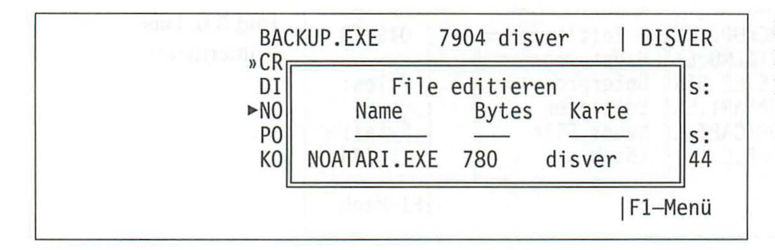


Bild 3.7: Das Editierfenster der Dateiliste

Dieser Menüpunkt erlaubt Ihnen, die einzelnen Dateien in der Liste zu editieren. Wenn Sie diesen Punkt anwählen, erscheint ein neues Fenster mit der ersten markierten Datei. Sie können nun nacheinander den Dateinamen, die Dateilänge und den Kartennamen ändern. Durch Bestätigen mit [Return] im Feld des Kartennamens wird, falls vorhanden, die nächste markierte Datei zum Editieren angezeigt.

Neues File
Neue Dateien können nicht nur von der Karte eingelesen, sondern auch manuell eingegeben werden. Es erscheint das gleiche Fenster wie beim Menüpunkt *Editieren*, allerdings mit leeren Eingabefeldern. Wenn Sie die Daten der neuen Datei eingegeben haben, wird sie an das Ende der Dateiliste gehängt.

Löschen
Sie löschen mit diesem Menüpunkt alle markierten Dateien. **Achtung!** Diese Operation läßt sich nicht rückgängig machen! Die gelöschten Dateien werden nicht zwischengepuffert!

Optionsmenü

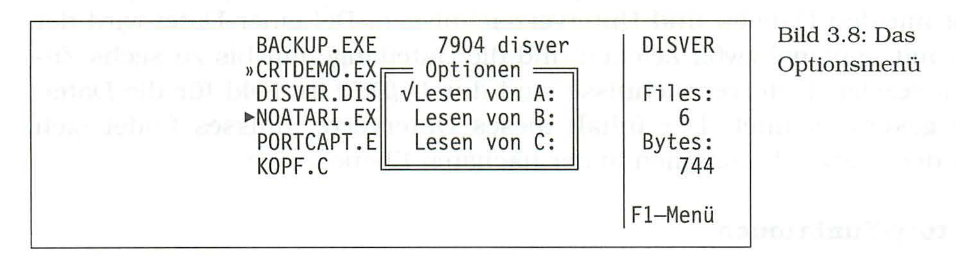


Bild 3.8: Das Optionsmenü

Mit Hilfe der Menüpunkte

- *Lesen von A:*
- *Lesen von B:*
- *Lesen von C:*

können Sie auswählen, von welchem Laufwerk gelesen wird. Auch hier wird nicht auf das Vorhandensein der Laufwerke geprüft, speziell trifft das für die Laufwerke A: und B: zu.

Directory-Bildschirm
Was nützt ein Verwaltungsprogramm, wenn man den Directory-Inhalt nicht lesen kann? Nachdem Sie aus dem Projektmenü den Punkt *Karte einlesen* gewählt haben, wird der Directory-Bildschirm aufgebaut und das komplette Inhaltsverzeichnis der Karte im gewählten Laufwerk eingelesen. Anschließend wird der Name der Karte angezeigt, der auch editiert werden kann. Danach erscheint die Liste der Directory-Einträge, und Sie können mit der Bearbeitung beginnen. Anhand des folgenden Bilds soll die Anzeige der Directory-Liste erklärt werden.

```
                    ──────────── │(H)auptprg.
        Level: 0                 │(U)nterprg.
                                 │(Ü)bernehmen
        Dir: A:                  │(E)ditieren
        - BACKUP.EXE       7904  │(A)lle
        - CRTDEMO.EXE     13776  │(S)torno
        - DISVER.DIS       3893  │(N)ächste
        - NOATARI.EXE       780  │(B)eenden
```

Bild 3.9: Der Directory-Bildschirm

Das komplette Directory wird ebenenweise eingelesen. Das Root- oder Basisverzeichnis gilt als Level 0, die darin enthaltenen Unterverzeichnisse als Level 1, die Unterverzeichnisse innerhalb der Verzeichnisse von Level 1 als Level 2 und so weiter. Zu Beginn jeder Ebene wird als Markierung ein Trennstrich und in der nächsten Zeile die Nummer der Ebene eingefügt. Durch eine Leerzeile getrennt folgt der Name des Verzeichnisses und sein Inhalt mit den Dateien und Unterverzeichnissen. Bei einer Datei wird der Name mit maximal zwölf Zeichen und die Dateilänge mit bis zu sechs Ziffern angezeigt. Unterverzeichnisse sind durch *(DIR)* im Feld für die Dateilänge gekennzeichnet. Der Inhalt dieses Unterverzeichnisses findet sich unter dem Verzeichnisnamen in der nächsten Ebene.

Directory-Funktionen

Zur Bearbeitung dieser Liste stehen folgende Funktionen zur Verfügung:

- *Hauptprogramm*
- *Unterprogramm*
- *Übernehmen*
- *Editieren*
- *Alle*
- *Storno*
- *Nächste*
- *Beenden*

Wählen Sie eine der Funktionen durch Drücken des eingeklammerten Buchstabens an. Wie beim Editiermenü des Dateibildschirms müssen auch bei den meisten der obigen Funktionen einer oder mehrere Einträge der Liste markiert sein. Die Markierung geschieht, genau wie vorher, durch Drücken der Leertaste, wenn der Cursor in der gewünschten Zeile steht. Auch hier wird die Markierung optisch dadurch gekennzeichnet, daß in der ersten Spalte der Zeile der Minusstrich durch ein nach rechts zeigendes Dreieck (ASCII-Code 16) ersetzt wird. Eine Besonderheit ist das Markieren der Directory-Überschrift. Dies hat die Markierung des kompletten Inhalts des Verzeichnisses zur Folge.

Die Funktionen *Hauptprogramm* und *Unterprogramm* haben die gleiche Wirkung wie im Dateibildschirm. Durch *Übernehmen* werden die markierten Einträge in die Dateiliste übernommen und in der Directory-Liste gelöscht. Bei *Editieren* erscheint ein ähnliches Fenster wie zuvor, das das Ändern des Dateinamens ermöglicht.

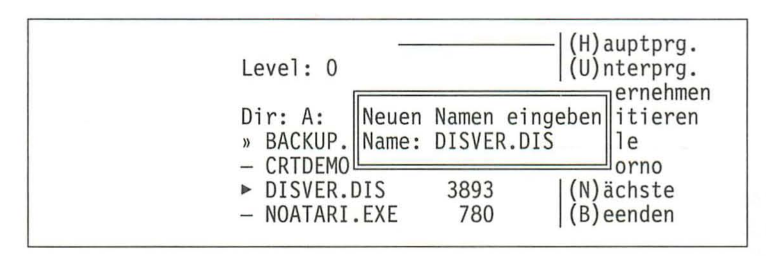


Bild 3.10: Das Editierfenster im Directory-Bildschirm

Der Punkt *Alle* markiert alle Einträge der Directory-Liste. Das Aufheben aller Markierungen ist durch den Punkt *Storno* möglich. *Nächste* liest das komplette Verzeichnis der nächsten Karte ein. Haben Sie alle Karten eingelesen, kehren Sie mit dem Punkt *Beenden* zum Dateibildschirm zurück. Die neuen Einträge wurden an das Ende der Dateiliste gehängt.

3.2.2 Das Listing des Beispielprogramms

Im folgenden finden Sie das recht umfangreiche Listing. Wie schon zu Beginn gesagt, mußten aus Platzgründen im Listing Leerzeilen herausgenommen und mehrere Befehle auf eine Zeile geschrieben werden.

```
/* —— DISVER PORTFOLIO ————————————————
 *
 *  BESCHREIBUNG:  DISVER PORTFOLIO stellt ein universelles
 *                 Hilfsmittel zum Verwalten der Portfolio-RAM-
 *                 Karten.
 *
 *  BEMERKUNGEN:   DISVER PORTFOLIO ist eine abgemagerte Version
 *                 der Programme DISVER AMIGA und DISVER PC, die
 *                 auf den jeweiligen Rechnern auch die Verwaltung
 *                 der Disketteninhalte erleichtern.
 *
 *  ÄNDERUNGEN:    Version 1.0    90/08/07    schuschk
 *
 *  INCLUDE-DATEIEN:
 */
#include <alloc.h>
#include <bios.h>
#include <mem.h>
#include <io.h>
```

```
#include <stdlib.h>
#include <dir.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <fcntl.h>
#include <string.h>
#include <time.h>
#include "typedef.h"
/*
 *  DEFINES:
 */
/* —— globale Defines ————————————— */
/*#define DEBUG*/
#define MAXFILES        500
#define MAXDIRS         200
#define MAXDIRSTACKS    50
#define FILES           0
#define DIR             1
#define DIRSTACK        2
/* —— Defines für den File-Requester ———————— */
#define MO_LESEN        0
#define MO_SCHREIBEN    1
/* —— Defines für die Scroll-Routinen ———————— */
#define HOME             0
#define PAGEUP          -8
#define UP              -1
#define DOWN             1
#define PAGEDOWN         8
#define END              999
/*
 *  VARIABLEN:
 */
/* —— Menü-Definitionen ——————————————— */
static CHAR *MainMenu[] = { "psedo",
    " Hauptmenü ", "Projekt", "Sortieren", "Editieren", "Drucken",
    "Optionen"};
static CHAR *ProjectMenu[] = { "nlsküe",
    " Projekt ", "Neu", "Laden", "Speichern", "Karte einlesen",
    "Über", "Ende"};
static CHAR *SortMenu[] = { "nbk",
    " Sortieren ", "nach Namen", "nach Bytes", "nach Karte"};
static CHAR *EditMenu[] = { "huenl",
    " Editieren ", "Hauptprogramm", "Unterprogramm", "Editieren",
    "Neues File", "Löschen"};
static CHAR *OptionsMenu[] = { "abc",
    " Optionen ", "√Lesen von A:", " Lesen von B:",
    " Lesen von C:"};

/* —— Variable für die Statusmeldungen ——————— */
CHAR    *PrintAllMsg = "Alle Daten werden gedruckt...",
        *DataLoadMsg = "Daten werden geladen...",
        *DataSaveMsg = "Daten werden gespeichert...",
        *SortMsg     = "Sortiervorgang läuft...",
```

```
        *DirReadMsg  = "Directory wird eingelesen...";
CHAR    *pcStatusWinPtr;

/* —— Variable für die Fehlermeldungen ———————— */
CHAR    *DirErr             = "!! Directory existiert nicht !!",
        *NoPlaceErr         = "!! Kein Platz für mehr Einträge !!",
        *NoFileErr          = "!! Datei nicht vorhanden !!",
        *NotADisverFileErr  = "!! Keine DISVER-Datei !!",
        *FileNotOpenErr     = "!! Kann Datei nicht öffnen !!",
        *FileNotSavedErr    = "!! Kann Daten nicht schreiben !!",
        *DriveNotReadyErr   = "!! Keine Karte eingesteckt !!",
        *NoFileClickedErr   = "!! Kein File markiert !!",
        *NoMemoryErr        = "!! Kein Speicher mehr frei !!",
        *DiskNotReadableErr = "!! Karte kann nicht gelesen werden !!",
        *PrinterNotReadyErr = "!! Drucker nicht ansprechbar !!";

/* —— globale Variable ———————————————— */
/* Struktur der File-Liste */
struct FileEntries {
    CHAR    acName[13]; /* Name */
    LONG    lBytes;     /* Länge in Bytes */
    CHAR    cSelect,    /* Selektierung */
            acDisk[11]; /* Karte, auf der sich das File befindet */
};
struct FileEntries *File[MAXFILES];
/* Struktur der Directory-Liste */
struct DirEntries {
    CHAR    acName[28]; /* Name des Directory-Eintrags */
    LONG    lBytes;     /* Länge in Bytes */
    CHAR    cSelect;    /* Markierungsflag */
};
struct DirEntries *Dir[200];
/* Struktur der Liste des Directory-Stapels */
struct DirStackEntries {
    CHAR    acName[128];    /* Pfad der Directory */
    INT     iLevel;         /* Grad der Verschachtelung */
};
struct DirStackEntries *DirStack[50];
BYTE    bModify = 0,        /* Änderungen-Flag */
        bEnde = 0,          /* Ende-Flag im File-Fenster */
        bEnde1, bEnde2;     /* Ende-Flags im Disk-Fenster */
CHAR    acDiskName[13],     /* Name der eingelesenen Karte */
        acDrive[4] = "A:\\",    /* LW, dessen Dir. gelesen wird */
        acDirName[31];      /* Name des einzulesenden Verz. */
INT     iFilesTop = -1,     /* ob. Grenze der File-Einträge */
        iDirTop = -1,       /* obere Grenze der Directory-Eintr. */
        iDirStackTop = -1,  /* obere Grenze der Dir.stapel-Eintr. */
        iDirPtr = 0,        /* Erster angezeigter Dir-Eintrag */
        iDirStackAnz = 0,   /* Anzahl Directories im Stapel */
        iDirStackPtr = 0,   /* Zeiger auf einzulesende Directory */
        iDirAnz = -0,       /* Anzahl eingel. Directory-Einträge */
        iLevel = 0,         /* aktuelle Verschachtelungsebene */
        iFileAnz = -1,      /* Anzahl gespeicherter Files */
        iFilePtr,           /* Nummer des ersten angez. Files */
```

```
        iHauptPrgPtr = -1,  /* Nummer der markierten Hauptprg. */
        iDirCursorPos = 0,  /* Cursor-Position im Dir-Fenster */
        iFileCursorPos = 0; /* Cursor-Position im File-Fenster */
LONG    lHauptPrgBytes;     /* Bytesumme des Hauptprogrammes */
union REGS  Regs;           /* Registersatz */
/* ------------------------------------ */
/* --- Fensterhintergrund retten ------------ */
CHAR    *StoreWindow(iLeft, iTop, iWidth, iLength)
INT     iLeft, iTop,        /* XY-Koordinaten der ob. linken Ecke */
        iWidth, iLength;    /* Breite und Höhe des Fensters */
{
    CHAR      *pcPtr, *pcBufPtr;    /* Zeiger auf Fensterpuffer */
    CHAR far *pcScrPtr;             /* Zeiger in Bildschirmsp. */
    INT       i, j,                 /* Zähler */
              iSize;                /* Größe des Puffers */
    pcPtr = NULL;

    /* Größe des Bereiches, dabei 4 Bytes für Koordinaten */
    iSize = 4 + (2 * iLength * iWidth);
    /* Puffer belegen und Zeiger retten */
    pcPtr = malloc(iSize);
    pcBufPtr = pcPtr;
    /* Koordinaten in den Puffer bringen */
    *pcBufPtr++ = (CHAR)iLeft; *pcBufPtr++ = (CHAR)iTop;
    *pcBufPtr++ = (CHAR)iWidth; *pcBufPtr++ = (CHAR)iLength;
    /* Fensterinhalt sichern */
    for(i = 0; i < iLength; i++){
        pcScrPtr = MK_FP(0xB000, ((iTop + i) * 160) + (iLeft * 2));
        for(j = 0; j < (2 * iWidth); j++) *pcBufPtr++ = *pcScrPtr++;
    }
    return(pcPtr);
}
/* --- Fenster schließen ------------------- */
void    CloseWindow(pcPtr)
CHAR    *pcPtr;            /* Zeiger auf Fensterpuffer */
{
    CHAR     *pcBufPtr,            /* Zeiger in Fensterpuffer */
             cLeft, cTop,          /* linke obere Ecke des Fensters */
             cWidth, cLength,      /* Breite und Länge des Fensters */
             i, j;                 /* Zähler */
    CHAR far *pcScrPtr;            /* Zeiger in Bildschirmspeicher */

    /* Pufferzeiger retten */
    pcBufPtr = pcPtr;
    /* Koordinaten aus dem Puffer holen */
    cLeft = *pcBufPtr++; cTop = *pcBufPtr++;
    cWidth = *pcBufPtr++; cLength = *pcBufPtr++;
    /* Fensterinhalt zurückschreiben */
    for(i = 0; i < cLength; i++){
        pcScrPtr = MK_FP(0xB000, ((cTop + i) * 160) + (cLeft * 2));
        for(j = 0; j < (2 * cWidth); j++) *pcScrPtr++ = *pcBufPtr++;
    }
    /* Pufferspeicher wieder freigeben */
    free(pcPtr);
```

```
    /* Bildschirm refreshen */
#ifndef DEBUG
    Regs.x.ax = 0x1200;
    int86(97, &Regs, &Regs);
#endif
    return;
}
/* --- Rechteckigen Bildschirmbereich löschen --------- */
void    ClearArea(iLeft, iTop, iWidth, iLength)
INT     iLeft, iTop,          /* linke obere Ecke des Bereiches */
        iWidth, iLength;      /* Breite und Länge (Höhe) */
{
    INT far     *piPtr;       /* Zeiger in Bildschirmspeicher */
    INT         i, j;         /* Zähler */

    /* Bereich löschen */
    for(i = 0; i < iLength; i++){
        piPtr = (INT far *)MK_FP(0xB000, ((iTop+i)*160)+(iLeft*2));
        for(j = 0; j < iWidth; j++) *piPtr++ = 0x0720;
    }
    /* Bildschirm refreshen */
#ifndef DEBUG
    Regs.x.ax = 0x1200;
    int86(97, &Regs, &Regs);
#endif
    return;
}
/* --- Doppelten Rahmen zeichnen ---------- */
void    DrawBorder(iLeft, iTop, iWidth, iLength)
INT     iLeft,  iTop,         /* linke obere Ecke des Rahmens */
        iWidth, iLength;      /* Breite und Höhe des Rahmens */
{
    INT far *piPtr;           /* Zeiger in Bildschirmspeicher */
    INT     i;                /* Zähler */

    piPtr = (INT far *)MK_FP(0xB000, (iTop * 160) + (iLeft * 2));
    *piPtr++ = 0x07c9;
    for(i = 0; i <= (iWidth-3); i++) *piPtr++ = 0x07cd;
    *piPtr = 0x07bb; piPtr += 80;
    for(i = 0; i <= (iLength-3); i++, piPtr += 80) *piPtr = 0x07ba;
    *piPtr-- = 0x07bc;
    for(i = 0; i <= (iWidth-3); i++) *piPtr-- = 0x07cd;
    *piPtr = 0x07c8; piPtr -= 80;
    for(i = 0; i <= (iLength-3); i++, piPtr -= 80) *piPtr = 0x07ba;
    /* Bildschirm refreshen */
#ifndef DEBUG
    Regs.x.ax = 0x1200;
    int86(97, &Regs, &Regs);
#endif
    return;
}
/* --- Fenster öffnen -------------------- */
CHAR    *OpenWindow(iLeft, iTop, iWidth, iLength)
INT     iLeft, iTop,          /* linke obere Ecke des Fensters */
```

```
        iWidth, iLength;     /* Breite und Höhe des Fensters */
{
    CHAR    *pcPtr;

    /* Hintergrund retten */
    pcPtr = StoreWindow(iLeft, iTop, iWidth, iLength);
    if(pcPtr != NULL){
        /* Hintergrund löschen */
        ClearArea(iLeft, iTop, iWidth, iLength);
        /* Rahmen zeichnen */
        DrawBorder(iLeft, iTop, iWidth, iLength);
    }
    return(pcPtr);
}
/* —— Taste holen und ASCII- sowie ——————————
   —— erweiterten Tastaturcode zurückgeben ———————— */
void    GetKey(pcKey, pcKeyExt)
CHAR    *pcKey,      /* ASCII-Code */
        *pcKeyExt;   /* erweiterter Tastaturcode */
{
    Regs.h.ah = 0x0;
    int86(0x16, &Regs, &Regs);
    *pcKey = Regs.h.al; *pcKeyExt = Regs.h.ah;
    return;
}
/* —— Eingabe-Routine ——————————————— */
INT      Input(iXPos, iYPos, iWidth, pcString)
INT      iXPos, iYPos,       /* linke Ecke des Eingabefeldes */
         iWidth;             /* Breite des Eingabefeldes */
CHAR     *pcString;          /* Zeiger auf String (Vorgabe möglich) */
{
    BYTE    bEnde;           /* Ende-Flag */
    INT     iEscape,         /* Abbruch-Flag */
            iRelCursor,      /* Cursor-Position im String */
            iLength;         /* aktuelle Länge der Eingabe */
    CHAR    *pcText,         /* Eingabepuffer */
            *pcPreset,       /* Vorgabepuffer */
            cKey, cKeyExt;   /* ASCII- und erweiterter Tastencode */

    /* Pufferspeicher für die Vorgabe und die Eingabe reservieren */
    pcText = malloc(iWidth + 2); pcPreset = malloc(iWidth + 2);
    /* Vorgabe retten */
    strcpy(pcPreset, pcString);
    /* Cursor auf Koordinaten setzen */
    iRelCursor = 1;
    gotoxy(iXPos + iRelCursor - 1, iYPos);
    /* Vorgabe in Eingabepuffer kopieren */
    strcpy(pcText, pcString); strcat(pcText, " ");
    /* Platz freimachen */
    printf("%*s", iWidth + 1, " ");
    /* Eingabeschleife */
    bEnde = 0; iEscape = 0;
    do{
        iLength = strlen(pcText);
```

```
/* aktuellen Stand ausgeben */
gotoxy(iXPos, iYPos); printf("%-*s", iWidth, pcText);
gotoxy(iXPos + iRelCursor - 1, iYPos);
/* Taste holen */
GetKey(&cKey, &cKeyExt);
/* Auswerten */
if(cKey == 0){
    /* Sondertaste gedrückt */
    switch(cKeyExt){
        /* Cursor links */
        case 75:    if(iRelCursor > 1) iRelCursor–; break;
        /* Cursor rechts */
        case 77:    if(iRelCursor < iLength) iRelCursor++;
                    break;
        /* Home oder Pos1 */
        case 71:    iRelCursor = 1; break;
        /* Ende */
        case 79:    iRelCursor = iLength; break;
        /* Delete oder Entf */
        case 83:    if(iLength > 1){
                        memcpy(pcText + iRelCursor - 1,
                            pcText + iRelCursor,
                            iLength - iRelCursor + 1);
                     }
                     break;
        default:     break;
    }
}
else{
    /* ASCII-Taste gedrückt */
    switch(cKey){
        /* Escape */
        case 27:    strcpy(pcText, pcPreset);
                    bEnde = 1; iEscape = 1;
                    break;
        /* Backspace */
        case 8:     if((iLength > 1)&&(iRelCursor > 1)){
                        memcpy(pcText + iRelCursor - 2,
                            pcText + iRelCursor - 1,
                            iLength - iRelCursor + 2);
                        iRelCursor–;
                    }
                    break;
        /* Return */
        case 13:    *(pcText + strlen(pcText) - 1) = 0;
                    bEnde = 1; break;
        /* normales Textzeichen */
        default:    if ((cKey > 31) && (iLength <= iWidth)){
                        memmove(pcText + iRelCursor,
                            pcText + iRelCursor - 1,
                            iLength - iRelCursor + 2);
                        *(pcText + iRelCursor - 1) = cKey;
                        iRelCursor++;
                    }
```

```
                        break;
            }
        }
    }while(bEnde == 0);
    /* Ergebnis anzeigen */
    gotoxy(iXPos, iYPos); printf("%-*s", iWidth, pcText);
    /* Eingabepuffer kopieren */
    strcpy(pcString, pcText);
    /* Speicher wieder freigeben */
    free(pcPreset); free(pcText);
    return(iEscape);
}
/* —— Fehlermeldung ausgeben ——————————— */
void    Error(iErrNum)
INT     iErrNum;        /* Fehlernummer */
{
    CHAR    *pcErrMsg,      /* Zeiger auf die Fehlermeldung */
            *pcWinPtr;      /* Zeiger auf Fensterpuffer */
    INT     iLeft, iTop,    /* Dimensionen des Fensters */
            iWidth, iLength;

    /* Fehlermeldung holen */
    switch(iErrNum){
        case 2:     pcErrMsg = DirErr; break;
        case 3:     pcErrMsg = NoPlaceErr; break;
        case 4:     pcErrMsg = NoFileErr; break;
        case 5:     pcErrMsg = NoPlaceErr; break;
        case 6:     pcErrMsg = NotADisverFileErr; break;
        case 7:     pcErrMsg = FileNotOpenErr; break;
        case 8:     pcErrMsg = FileNotSavedErr; break;
        case 9:     pcErrMsg = PrinterNotReadyErr; break;
        case 11:    pcErrMsg = DriveNotReadyErr; break;
        case 14:    pcErrMsg = NoFileClickedErr; break;
        case 15:    pcErrMsg = NoMemoryErr; break;
        case 16:    pcErrMsg = DiskNotReadableErr; break;
        default:    pcErrMsg = NULL; break;
    }
    if(pcErrMsg == NULL) return;
    /* Breite und Höhe des Fensters festlegen */
    iWidth = strlen(pcErrMsg) + 4; iLength = 3;
    /* Spalte und Zeile der oberen linken Ecke bestimmen */
    iLeft = (40 - iWidth) / 2; iTop = (8 - iLength) / 2;
    /* Fenster aufmachen */
    pcWinPtr = OpenWindow(iLeft, iTop, iWidth, iLength);
    if(pcWinPtr == NULL) return;
    /* Fehlermeldung ausgeben */
    gotoxy(iLeft + 3, iTop + 2); printf("%s", pcErrMsg);
    /* Taste holen und Fenster schließen */
    while(!kbhit()); getchar();
    CloseWindow(pcWinPtr);
    return;
}
/* —— Statusmeldung ausgeben ——————————— */
void    Status(iStatus)
```

```
INT     iStatus;         /* Nummer der Statusmeldung */
{
    CHAR    *pcStatusMsg;   /* Zeiger auf die Statusmeldung */
    INT     iLeft, iTop,    /* Dimensionen des Fensters */
            iWidth, iLength;

    /* evtl. Statusmeldung entfernen */
    if((iStatus == 0) && (pcStatusWinPtr != NULL))
        CloseWindow(pcStatusWinPtr);
    /* Statusmeldung zuweisen */
    switch(iStatus){
        case 200:   pcStatusMsg = PrintAllMsg; break;
        case 203:   pcStatusMsg = DataLoadMsg; break;
        case 204:   pcStatusMsg = DataSaveMsg; break;
        case 205:   pcStatusMsg = SortMsg; break;
        case 206:   pcStatusMsg = DirReadMsg; break;
        default:    pcStatusMsg = NULL; break;
    }
    if(pcStatusMsg == NULL) return;
    /* Breite und Höhe des Fensters festlegen */
    iWidth = strlen(pcStatusMsg) + 4; iLength = 3;
    /* Spalte und Zeile der oberen linken Ecke bestimmen */
    iLeft = (40 - iWidth) / 2; iTop = (8 - iLength) / 2;
    /* Fenster aufmachen */
    pcStatusWinPtr = OpenWindow(iLeft, iTop, iWidth, iLength);
    if(pcStatusWinPtr == NULL) return;
    /* Fehlermeldung ausgeben */
    gotoxy(iLeft + 3, iTop + 2);
    printf("%s", pcStatusMsg);
    return;
}
/* —— Speicherverwaltungsroutine ———————— */
INT      AllocEntry(iPtr, iChoice)
INT      iPtr,      /* Position für die Speicher angefordert wird */
         iChoice;   /* Array, für das der Speicher sein soll */
{
    switch(iChoice){
        case FILES:
            if (iPtr > iFilesTop && iPtr <= MAXFILES){
                /* Speicher anfordern */
                File[iPtr] = (struct FileEntries *)calloc(1,
                    sizeof(struct FileEntries));
                /* kein Speicher mehr frei ? */
                if (File[iPtr] == NULL){
                    Error(5); return(1);
                }
                /* obere Grenze erhöhen */
                iFilesTop++;
            }
            return(0);
        case DIR:
            if (iPtr > iDirTop && iPtr <= MAXDIRS){
                /* Speicher anfordern */
                Dir[iPtr] = (struct DirEntries *)calloc(1,
```

```
                    sizeof(struct DirEntries));
                /* kein Speicher mehr frei ? */
                if (Dir[iPtr] == NULL){
                    Error(5); return(1);
                }
                /* obere Grenze erhöhen */
                iDirTop++;
            }
            return(0);
        case DIRSTACK:
            if (iPtr > iDirStackTop && iPtr <= MAXDIRSTACKS){
                /* Speicher anfordern */
                DirStack[iPtr] = (struct DirStackEntries *)calloc(1,
                    sizeof(struct DirStackEntries));
                /* kein Speicher mehr frei ? */
                if (DirStack[iPtr] == NULL){
                    Error(5); return(1);
                }
                /* obere Grenze erhöhen */
                iDirStackTop++;
            }
            return(0);
        default:
            return(1);
    }
}
/* —— Directory-Bildschirm aufbauen ————————— */
void    SetupDiskScreen()
{
    clrscr();
    printf("                              |(H)auptprg.\n");
    printf("                              |(U)nterprg.\n");
    printf("                              |(Ü)bernehmen\n");
    printf("                              |(E)ditieren\n");
    printf("                              |(A)lle\n");
    printf("                              |(S)torno\n");
    printf("                              |(N)ächste\n");
    printf("                              |(B)eenden");
    return;
}
/* —— Sortieren der Directory-Einträge im Bereich ——— */
/* —— von iAnfang bis iEnde ——————————— */
void    DirSort(iAnfang, iEnde)
INT     iAnfang, iEnde;
{
    INT     i, j;                   /* Zähler */
    struct DirEntries   *SwapPtr;   /* scratch pointer */

    for (i = iAnfang; i <= iEnde - 1; i++){
        for (j = i + 1; j <= iEnde; j++){
            if (strcmp(Dir[i]->acName, Dir[j]->acName) > 0){
                /* Zeiger auf die Strukturen vertauschen */
                SwapPtr = Dir[i]; Dir[i] = Dir[j]; Dir[j] = SwapPtr;
            }
```

```
        }
    }
    return;
}
/* —— Directory "pcVerzeichnis" einlesen, ab Position —— */
/* —— "iCounter" in die Liste der Directory-Einträge ——— */
/* —— bringen und Endwert zurückgeben ——————— */
INT     LeseDir(pcVerzeichnis, iCounter)
CHAR    *pcVerzeichnis;
INT     iCounter;
{
    struct ffblk    Eintrag;    /* Dateiinfo-Struktur */
    INT             iSuccess;   /* Funktionsergebnis */

    /* Namen des aktuellen und übergeordneten Verz. ermitteln */
    strcpy(acDirName, pcVerzeichnis);
    /* Abschließenden Backslash entfernen */
    *(acDirName + strlen(acDirName) - 1) = 0;
    /* Speicherplatz reservieren */
    if (AllocEntry(iCounter, DIR) == 1) return(iCounter);
    /* Namen des aktuellen Verzeichnisses isolieren */
    if (iLevel > 1)
        sprintf(Dir[iCounter]->acName, "Dir: %-s",
            strrchr(acDirName,'\\'));
    else
        sprintf(Dir[iCounter]->acName, "Dir: %-s", acDirName);
    /* Directory einlesen */
    Dir[iCounter]->lBytes = 0;
    strcat(acDirName, "\\*.*");
    iSuccess = findfirst(acDirName, &Eintrag, 0x37);
    while(iSuccess == 0){
        /* Alle Einträge bis auf "." und ".." einlesen */
        if ((strcmp(Eintrag.ff_name, "."))
            && (strcmp(Eintrag.ff_name, ".."))){
            iCounter++;
            /* Speicherplatz reservieren */
            if (AllocEntry(iCounter, DIR) == 1){
                iSuccess = -1; iCounter-;
                break;
            }
            sprintf(Dir[iCounter]->acName, "- %-s",Eintrag.ff_name);
            if (Eintrag.ff_attrib & 0x10)
                /* Eintrag ist ein Unterdirectory */
                Dir[iCounter]->lBytes = -1;
            else
                /* Eintrag ist ein File */
                Dir[iCounter]->lBytes = Eintrag.ff_fsize;
        }
        iSuccess = findnext(&Eintrag);
    }
    return(iCounter);
}
/* —— Kopf für eine neue Verschachtelungsebene ———— */
void    LevelKopf()
```

```
{
    if (AllocEntry(iDirAnz, DIR) == 1) return;
    strcpy(Dir[iDirAnz]->acName, "—————————");
    Dir[iDirAnz]->lBytes = -3; iDirAnz++;
    if (AllocEntry(iDirAnz, DIR) == 1) return;
    sprintf(Dir[iDirAnz]->acName, "Level: %ld", iLevel);
    Dir[iDirAnz]->lBytes = -2; iDirAnz++;
    if (AllocEntry(iDirAnz, DIR) == 1) return;
    strcpy(Dir[iDirAnz]->acName, "                    ");
    Dir[iDirAnz]->lBytes = -4; iDirAnz++;
    return;
}
/* —— Kompletten Directory-Baum des Laufwerkes einlesen —— */
/* —— und die Anzahl der Einträge zurückgeben ——————— */
INT     DiskIn()
{
    INT     iAnz,            /* gelesene Einträge eines Unterverz. */
            iDummy,          /* Funktionsergebnis */
            i;               /* Zähler */
    BYTE    bDrv;            /* Laufwerksnummer (0=A:, 1=B:, ...) */
    CHAR    acBuffer[512],   /* Lesepuffer */
            *pcPtr;          /* scratch pointer */
    struct ffblk Eintrag;    /* Dateiinfoblock */

    /* Laufwerksnummer setzen */
    bDrv = acDrive[0] - 65;
    /* Testen, ob Laufwerk bereit */
    iDummy = absread(bDrv, 1, 1, acBuffer);
    if (iDummy == -1){
        Error(11); return(iDirAnz - 1);
    }
    Status(206);
    /* Diskettennamen lesen */
    strcpy(acBuffer, acDrive);
    iDummy = findfirst(acBuffer, &Eintrag, FA_LABEL);
    if (iDummy == 0){
        strcpy(acDiskName, Eintrag.ff_name);
        /* evtl. Punkt aus dem Namen entfernen */
        pcPtr = strchr(acDiskName, 46);
        if (pcPtr != NULL){
            while(*pcPtr != '\0'){
                *pcPtr = *(pcPtr + 1); pcPtr++;
            }
        }
    }
    else
        strcpy(acDiskName, "");
    LevelKopf();
    /* Speicherplatz reservieren */
    if (AllocEntry(0, DIRSTACK) == 1) return(iDirAnz - 1);
    strcpy(DirStack[0]->acName, acDrive);
    /* Directory-Baum einlesen */
    while (iDirStackPtr <= iDirStackAnz){
        if (DirStack[iDirStackPtr]->iLevel < 10){
```

```
        if (DirStack[iDirStackPtr]->iLevel > iLevel){
            iLevel++; LevelKopf();
        }
        /* Directory einlesen und sortieren */
        iAnz = LeseDir(DirStack[iDirStackPtr]->acName, iDirAnz);
        DirSort(iDirAnz + 1, iAnz);
        if (iAnz > 0){
            /* Diskettennamen in DirStack schreiben */
            if (iDirStackPtr == 0){
                strcpy(DirStack[0]->acName, acDrive);
                DirStack[0]->iLevel = 0;
            }
            /* Nach Unterdirectories suchen */
            for (i = iDirAnz + 1; i <= iAnz; i++){
                if (Dir[i]->lBytes == -1){
                    iDirStackAnz++;
                    if (AllocEntry(iDirStackAnz, DIRSTACK)==1){
                        iDirStackAnz-;
                        break;
                    }
                    strcpy(DirStack[iDirStackAnz]->acName,
                        DirStack[iDirStackPtr]->acName);
                    strcat(DirStack[iDirStackAnz]->acName,
                        (Dir[i]->acName) + 2);
                    strcat(DirStack[iDirStackAnz]->acName,"\\");
                    DirStack[iDirStackAnz]->iLevel = iLevel + 1;
                }
            }
            /* Leerzeile einfügen */
            if (AllocEntry(iAnz + 1, DIR) == 1) break;
            strcpy(Dir[iAnz + 1]->acName, "");
            Dir[iAnz + 1]->lBytes = -4; iDirAnz = iAnz + 2;
        }
        else
            Error(2);
    }
    iDirStackPtr++;
  }
  Status(0);
  return(iDirAnz - 1);
}
/* -- Einen Eintrag aus der Directory-Liste in der ------ */
/* -- angegebenen Zeile ausgeben ---------------- */
void    ShowOneDirEntry(iNumber, iRow)
INT     iNumber,        /* lfd. Nummer des Eintrags */
        iRow;           /* Zeile, in der ausgegeben wird */
{
    CHAR    cMarker,            /* Markierung */
            acDirEntry[27];     /* Ausgabepuffer */

    if (iNumber >= iDirAnz){
        sprintf(acDirEntry, "%26s", " "); return;
    }
    switch(Dir[iNumber]->lBytes){
```

```
        /* Eintrag ist eine Leerzeile */
        case -4:    sprintf(acDirEntry, "%-26s", " "); break;
        /* Eintrag ist ein Trennstrich */
        case -3:    sprintf(acDirEntry,
                      "────────────");
                    break;
        /* Eintrag ist eine Level-Überschrift */
        case -2:    sprintf(acDirEntry, "%-26s",
                        Dir[iNumber]->acName);
                    break;
        /* Eintrag ist ein Unterdirectory */
        case -1:    sprintf(acDirEntry, "%-14.14s (DIR)      ",
                        Dir[iNumber]->acName);
                    break;
        /* Eintrag ist eine Directory-Überschrift */
        case 0:     sprintf(acDirEntry, "%-26s",
                        Dir[iNumber]->acName);
                    break;
        /* Eintrag ist ein File */
        default:    switch(Dir[iNumber]->cSelect){
                        case 1:  cMarker = 16; break;
                        case 2:  cMarker = 175; break;
                        default: cMarker = '-'; break;
                    }
                    sprintf(acDirEntry, "%-14.14s %6ld    ",
                        Dir[iNumber]->acName, Dir[iNumber]->lBytes);
                    *acDirEntry = cMarker;
                    break;
    }
    gotoxy(1, iRow); printf("%s", acDirEntry);
    return;
}
/* ── Elemente der Directory-Liste ab Position "iDirPtr" ── */
/* ── auf dem Bildschirm ausgeben ────────────── */
void    ShowEntries()
{
    INT     i;              /* Zähler */

    if (iDirPtr + 8 > iDirAnz) iDirPtr = iDirAnz - 7;
    if (iDirPtr < 0) iDirPtr = 0;
    /* Einträge ausgeben */
    for (i = 0; i < 8; i++){
        ShowOneDirEntry(iDirPtr + i, i + 1);
    }
    return;
}
/* ── Einträge der Directory-Liste ────────────── */
/* ── entsprechend dem Argument scrollen ──────── */
void    DirScroll(iNum)
INT     iNum;       /* Anzahl Zeilen, um die gerollt wird */
{
    INT     i;      /* Zähler */

    switch (iNum){
```

```
        case HOME:      iDirPtr = 0;
                        ShowEntries(); break;
        case END:       iDirPtr = iDirAnz - 7;
                        ShowEntries(); break;
        case DOWN:      if (iDirPtr < iDirAnz - 8){
                            iDirPtr++;
                            Regs.x.ax = 0x0601; Regs.x.cx = 0;
                            Regs.x.dx = 0x0719; Regs.h.bh = 0;
                            int86(0x10, &Regs, &Regs);
                            ShowOneDirEntry(iDirPtr + 7, 8);
                        }
                        break;
        case UP:        if (iDirPtr > 0){
                            iDirPtr--;
                            Regs.x.ax = 0x0701; Regs.x.cx = 0;
                            Regs.x.dx = 0x0719; Regs.h.bh = 0;
                            int86(0x10, &Regs, &Regs);
                            ShowOneDirEntry(iDirPtr, 1);
                        }
                        break;
        case PAGEUP:    for (i = 0; i < 8; i++){
                            if (iDirPtr > 0){
                                iDirPtr--;
                                Regs.x.ax = 0x0701; Regs.x.cx = 0;
                                Regs.x.dx = 0x0719; Regs.h.bh = 0;
                                int86(0x10, &Regs, &Regs);
                                ShowOneDirEntry(iDirPtr, 1);
                            }
                        }
                        break;
        case PAGEDOWN:  for (i = 0; i < 8; i++){
                            if (iDirPtr < iDirAnz - 8){
                                iDirPtr++;
                                Regs.x.ax = 0x0601; Regs.x.cx = 0;
                                Regs.x.dx = 0x0719; Regs.h.bh = 0;
                                int86(0x10, &Regs, &Regs);
                                ShowOneDirEntry(iDirPtr + 7, 8);
                            }
                        }
                        break;
        default:        iDirPtr += iNum;
                        ShowEntries(); break;
    }
    return;
}
/* -- Alle Selektierungen rückgängig machen -------- */
void    DirHighlightClear()
{
    INT     i;      /* Zähler */

    for (i = 0; i <= iDirAnz; i++) Dir[i]->cSelect = 0;
    return;
}
/* -- Diskettennamen anzeigen und evtl. editieren lassen -- */
```

```
void    TestDiskName()
{
    CHAR    *pcWinPtr;            /* Zeiger auf Fensterpuffer */

    /* Requester öffnen und neuen Kartennamen abfragen */
    pcWinPtr = OpenWindow(9, 2, 22, 4);
    gotoxy(11, 4); printf("Kartennamen eingeben");
    gotoxy(13, 5); printf("Name:");
    Input(19, 5, 10, acDiskName);
    CloseWindow(pcWinPtr);
    ShowEntries();
    return;
}
/* —— Markierten Eintrag zum Hauptprogramm machen ——— */
void    MainProgram()
{
    INT     i;      /* Zähler */

    /* evtl. vorherige Hauptprogrammarkierungen löschen */
    for (i = 0; i <= iDirAnz; i++)
        if (Dir[i]->cSelect == 2) Dir[i]->cSelect = 0;
    /* erstes markierten Eintrag suchen */
    for (i = 0; (Dir[i]->cSelect == 0) && (i <= iDirAnz); i++);
    /* Eintrag markieren und Hauptprogrammpointer übergeben */
    iHauptPrgPtr = i; Dir[i]->cSelect = 2;
    /* Byteanzahl für spätere Addition merken */
    lHauptPrgBytes = Dir[i]->lBytes;
    /* Markierung anzeigen */
    ShowEntries();
    return;
}
/* —— Bytes aller markierten Einträge addieren ——— */
void    SubProgram()
{
    INT     i, j;   /* Zähler */

    j = -1;
    if (iHauptPrgPtr != -1){
        for (i = 0; i <= iDirAnz; i++){
            if (Dir[i]->cSelect == 1){
                lHauptPrgBytes += Dir[i]->lBytes;
                Dir[iHauptPrgPtr]->lBytes = lHauptPrgBytes;
                bModify = 1;
            }
            else{
                j++;
                if (i == iHauptPrgPtr) iHauptPrgPtr = j;
                strcpy(Dir[j]->acName, Dir[i]->acName);
                Dir[j]->lBytes = Dir[i]->lBytes;
                Dir[j]->cSelect = 0;
            }
        }
        for (i = j + 1; i <= iDirAnz; i++) Dir[i]->cSelect = 0;
        iDirAnz = j;
```

```
        Dir[iHauptPrgPtr]->cSelect = 1;
        if (iDirAnz < 19)
            iDirPtr = 0;
        else
            iDirPtr = iHauptPrgPtr - 4;
        iHauptPrgPtr = -1;
        ShowEntries();
    }
    return;
}
/* —— Alle markierten Einträge in die Liste ———— */
/* —— der Files übernehmen ————————— */
void    AddEntry()
{
    INT     i, j;       /* Zähler */

    if (iHauptPrgPtr < 1){
        /* alle markierten Einträge übernehmen */
        j = 2;
        for (i = 3; i <= iDirAnz; i++){
            if (Dir[i]->cSelect > 0){
                if (iFileAnz < MAXFILES){
                    /* Speicherplatz reservieren */
                    iFileAnz++;
                    if (AllocEntry(iFileAnz, FILES) == 1){
                        iFileAnz-; break;
                    }
                    /* Inhalte übertragen */
                    strcpy(File[iFileAnz]->acName,
                        (CHAR *)(Dir[i]->acName) + 2);
                    File[iFileAnz]->lBytes = Dir[i]->lBytes;
                    strcpy(File[iFileAnz]->acDisk, acDiskName);
                    File[iFileAnz]->cSelect = 0;
                    bModify = 1;
                }
                else
                    Error(3);
            }
            else{
                j++;
                strcpy(Dir[j]->acName, Dir[i]->acName);
                Dir[j]->lBytes = Dir[i]->lBytes;
                Dir[j]->cSelect = Dir[i]->cSelect;
            }
        }
        iDirAnz = j;
    }
    if (iDirAnz < 8)
        iDirPtr = 0;
    else
        if (iDirPtr + 7 > iDirAnz) iDirPtr = iDirAnz - 8;
    ShowEntries();
    return;
}
```

```
/* —— Initialisierung der Variablen ———————— */
void    InitDirVars()
{
    iDirStackPtr = iLevel = iDirPtr = iDirCursorPos = 0;
    iDirAnz = iDirStackAnz = 0;
    bEnde1 = bEnde2 = 0;
    iHauptPrgPtr = -1;
    return;
}
/* —— Markierung mit der Space-Taste entsprechend ——— */
/* —— der Position auswerten ————————— */
void    ClickOnEntries(iNumber)
INT     iNumber;          /* Nummer des markierten Eintrags */
{
    INT     j;      /* Zähler */

    if (iNumber != iHauptPrgPtr){
        if (Dir[iNumber]->lBytes > 0){
            /* Eintrag ist ein File -> Markierung umschalten */
            if (Dir[iNumber]->cSelect == 1)
                Dir[iNumber]->cSelect = 0;
            else
                if (Dir[iNumber]->cSelect == 0)
                    Dir[iNumber]->cSelect = 1;
        }
        else{
            /* Eintrag ist Directory-Überschrift -> Markierungen
               innerhalb des Directories umschalten */
            if (Dir[iNumber]->lBytes == 0){
                for (j = iNumber + 1; Dir[j]->lBytes > -2; j++){
                    if ((Dir[j]->cSelect==0) && (Dir[j]->lBytes>0))
                        Dir[j]->cSelect = 1;
                    else
                        if (Dir[j]->cSelect==1) Dir[j]->cSelect = 0;
                }
            }
        }
    }
    ShowEntries();
    return;
}
/* —— Tastendruck auswerten ————————— */
void    ActionOnDiskKey(cKey, cKeyExt)
CHAR    cKey,             /* ASCII-Code der Taste */
        cKeyExt;          /* erweiterter Tastaturcode */
{
    CHAR    *pcWinPtr;          /* Zeiger auf Fensterpuffer */
    INT     i;                  /* Zähler */

    if(cKey != 0){
        switch(cKey){
            /* Leertaste - Markieren */
            case 32:    ClickOnEntries(iDirPtr + iDirCursorPos);
                        break;
```

```
        /* H - Markierten Eintrag zum Hauptprogramm machen */
        case 'h':
        case 'H':   MainProgram(); break;
        /* U - Markierte Einträge zum Hauptprogramm addieren */
        case 'u':
        case 'U':   SubProgram(); break;
        /* Ü - Markierte Einträge übernehmen */
        case 'ü': case 'Ü':
        case '+':   AddEntry(); break;
        /* E - Namen der markierten Einträge editieren */
        case 'e':
        case 'E':   pcWinPtr = OpenWindow(9, 2, 22, 4);
                    gotoxy(11, 4);
                    printf("Neuen Namen eingeben");
                    gotoxy(11, 5); printf("Name:");
                    for (i = 4; i <= iDirAnz; i++){
                        if (Dir[i]->cSelect == 1){
                            Input(17,5,12,(Dir[i]->acName)+2);
                            Dir[i]->cSelect = 0;
                        }
                    }
                    CloseWindow(pcWinPtr);
                    ShowEntries(); break;
        /* A - Alle Einträge markieren */
        case 'a':
        case 'A':   for (i = 2; i <= iDirAnz; i++)
                        if ((Dir[i]->lBytes > 0) &&
                            (i != iHauptPrgPtr))
                            Dir[i]->cSelect = 1;
                    ShowEntries(); break;
        /* S - Alle Markierungen rückgängig machen */
        case 's':
        case 'S':   for (i = 0; i <= iDirAnz; i++)
                        Dir[i]->cSelect = 0;
                    iHauptPrgPtr = -1;
                    lHauptPrgBytes = 0;
                    ShowEntries(); break;
        /* B - Bearbeitung beenden */
        case 'b':
        case 'B':   bEnde1 = 1;
        /* N - Nächste Magnetkarte einlesen */
        case 'n':
        case 'N':   bEnde2 = 1; iHauptPrgPtr = -1;
                    DirHighlightClear(); break;
        default:    break;
    }
}
else{
    switch(cKeyExt){
        /* Cursor nach oben */
        case 72:    if(iDirCursorPos == 0)
                        DirScroll(UP);
                    else
                        iDirCursorPos-;
```

```
                    break;
            /* Cursor nach unten */
            case 80:    if(iDirCursorPos == 7)
                            DirScroll(DOWN);
                        else
                            if (iDirPtr + iDirCursorPos < iDirAnz)
                                iDirCursorPos++;
                        break;
            /* Bild nach oben (PageUp) */
            case 73:    DirScroll(PAGEUP); break;
            /* Bild nach unten (PageDn) */
            case 81:    DirScroll(PAGEDOWN); break;
            /* Pos1 (Home) */
            case 71:    DirScroll(HOME); break;
            /* Ende (End) */
            case 79:    DirScroll(END); break;
            default:    break;
        }
    }
    return;
}
/* —— Magnetkarte einlesen ————————————— */
void    DisketteEinlesen()
{
    CHAR    cKey, cKeyExt;  /* Tastenpuffer */

    /* Bildschirm aufbauen */
    SetupDiskScreen();
    /* Eingabeschleife */
    while (bEnde1 == 0){
        InitDirVars();
        /* Directory des gewählten Laufwerkes einlesen */
        iDirAnz = DiskIn();
        /* Diskettennamen zur Änderung anzeigen */
        if (iDirAnz > 0) TestDiskName();
        iDirPtr = 0; ShowEntries();
        while (bEnde2 == 0){
            /* Cursor positionieren */
            gotoxy(1, iDirCursorPos + 1);
            /* Hat der User sich gerührt ? */
            GetKey(&cKey, &cKeyExt);
            ActionOnDiskKey(cKey, cKeyExt);
        }
    }
    bEnde1 = 0;
    return;
}
/* —— File-Bildschirm aufbauen ————————————— */
void    SetupFileScreen()
{
    clrscr();
    printf("                                | DISVER\n");
    printf("                                |       \n");
    printf("                                | Files:\n");
```

```
    printf("                                    |       \n");
    printf("                                    | Bytes:\n");
    printf("                                    |       \n");
    printf("                                    |       \n");
    printf("                                    |F1-Menü");
    gotoxy(0, 0);
    return;
}
/* —— Bytes summieren ——————————————— */
LONG    do_sum()
{
    INT     i;              /* Zähler */
    LONG    lSumBytes;      /* Summe der Bytes */
    lSumBytes = 0;

    for(i = 0; i <= iFileAnz; i++) lSumBytes += File[i]->lBytes;
    return(lSumBytes);
}
/* —— Alle Selektierungen aufheben ——————————— */
void    HighLightClear()
{
    INT     i;

    for(i = 0; i <= iFileAnz; i++) File[i]->cSelect = 0;
    return;
}
/* —— Einen Eintrag aus der Liste der Files in der ——— */
/* —— angegebenen Zeile ausgeben ——————————— */
void    ShowOneFileEntry(iNumber, iRow)
INT     iNumber,        /* lfd. Nummer des Eintrags */
        iRow;           /* Zeile, in der ausgegeben wird */
{
    CHAR    cMarker;        /* Markierung */

    gotoxy(1, iRow);
    if (iNumber > iFileAnz){
        printf("%31s", " ");
        return;
    }
    switch(File[iNumber]->cSelect){
        /* Normale Markierung */
        case 1:     cMarker = 16; break;
        /* als Hauptprogramm markiert */
        case 2:     cMarker = 175; break;
        /* nicht markiert */
        default:    cMarker = 32; break;
    }
    printf("%c%-12s %6ld %-10s", cMarker, File[iNumber]->acName,
        File[iNumber]->lBytes, File[iNumber]->acDisk);
    return;
}
/* —— Files ab Position 'iFilePtr' anzeigen ————— */
void    ShowFiles()
{
```

```
    INT     i;      /* Zähler */

    if (iFilePtr + 8 > iFileAnz) iFilePtr = iFileAnz - 7;
    if (iFilePtr < 0) iFilePtr = 0;
    for (i = 0; i < 8; i++) ShowOneFileEntry(iFilePtr + i, i + 1);
    return;
}
/* -- Liste der Files entsprechend dem Argument scrollen -- */
void    FileScroll(iNum)
INT     iNum;   /* Zeilenanzahl, um die gescrollt wird */
{
    INT     i;

    switch(iNum){
        case END:       iFilePtr = iFileAnz - 7;
                        ShowFiles(); break;
        case HOME:      iFilePtr = 0;
                        ShowFiles(); break;
        case DOWN:      if (iFilePtr < iFileAnz - 8){
                            iFilePtr++;
                            Regs.x.ax = 0x0601; Regs.x.cx = 0;
                            Regs.x.dx = 0x071e; Regs.h.bh = 0;
                            int86(0x10, &Regs, &Regs);
                            ShowOneFileEntry(iFilePtr + 7, 8);
                        }
                        break;
        case UP:        if (iFilePtr > 0){
                            iFilePtr-;
                            Regs.x.ax = 0x0701; Regs.x.cx = 0;
                            Regs.x.dx = 0x071e; Regs.h.bh = 0;
                            int86(0x10, &Regs, &Regs);
                            ShowOneFileEntry(iFilePtr, 1);
                        }
                        break;
        case PAGEUP:    for (i = 0; i < 8; i++){
                            if (iFilePtr > 0){
                                iFilePtr-;
                                Regs.x.ax = 0x0701; Regs.x.cx = 0;
                                Regs.x.dx = 0x071e; Regs.h.bh = 0;
                                int86(0x10, &Regs, &Regs);
                                ShowOneFileEntry(iFilePtr, 1);
                            }
                        }
                        break;
        case PAGEDOWN:  for (i = 0; i < 8; i++){
                            if (iFilePtr < iFileAnz - 8){
                                iFilePtr++;
                                Regs.x.ax = 0x0601; Regs.x.cx = 0;
                                Regs.x.dx = 0x071e; Regs.h.bh = 0;
                                int86(0x10, &Regs, &Regs);
                                ShowOneFileEntry(iFilePtr + 7, 8);
                            }
                        }
                        break;
```

```
        default:        iFilePtr += iNum;
                        ShowFiles(); break;
    }
    return;
}
/* —— Menü anzeigen und gewählten Menüpunkt zurückliefern — */
CHAR    Menu(apcMenuArray)
CHAR    *apcMenuArray[];        /* Zeiger auf Array der Menüoptionen */
{
    CHAR    cKey, cKeyExt,      /* Tasten */
            cChar,              /* Menüauswahl */
            *pcWinPtr;          /* Zeiger auf Fensterpuffer */
    INT     iLeft, iTop,        /* Dimensionen des Fensters */
            iWidth, iLength,
            iMenuCursorPos,     /* Zeile des Auswahlcursors */
            i;                  /* Zähler */

    /* Höhe des Fensters berechnen */
    iLength = strlen(apcMenuArray[0]);
    /* breitesten String suchen */
    iWidth = 0;
    for (i = 1; i <= iLength; i++)
        if(iWidth < strlen(apcMenuArray[i]))
            iWidth = strlen(apcMenuArray[i]);
    /* Rahmen in Breite und Höhe einbeziehen */
    iWidth += 4; iLength += 2;
    /* Spalte und Zeile der oberen linken Ecke bestimmen */
    iLeft = (40 - iWidth) / 2; iTop = (8 - iLength) / 2;
    /* Fenster aufmachen */
    pcWinPtr = OpenWindow(iLeft, iTop, iWidth, iLength);
    if(pcWinPtr == NULL) return(0);
    /* Menütitel zentriert ausgeben */
    gotoxy(iLeft+1+((iWidth-strlen(apcMenuArray[1]))/2), iTop+1);
    printf("%s", apcMenuArray[1]);
    /* Menüoptionen ausgeben */
    for(i = 2; i <= iLength - 1; i++){
        gotoxy(iLeft + 3, iTop + i); printf("%s", apcMenuArray[i]);
    }
    iMenuCursorPos = 0; cChar = 0;
    /* Taste holen und auswerten */
    do{
        gotoxy(iLeft + 3, iTop + 2 + iMenuCursorPos);
        GetKey(&cKey, &cKeyExt);
        if (cKey != 0){
            if (cKey == 27){
                cChar = 27; break;
            }
            if (cKey == 13)
                cChar = *(apcMenuArray[0] + iMenuCursorPos);
            else
                cChar = *strchr(apcMenuArray[0], cKey);
        }
        else{
            switch (cKeyExt){
```

```
                /* Cursor nach oben */
                case 72:    iMenuCursorPos--;
                            if (iMenuCursorPos < 0)
                                iMenuCursorPos = 0;
                            break;
                /* Cursor nach unten */
                case 80:    iMenuCursorPos++;
                            if (iMenuCursorPos > iLength - 3)
                                iMenuCursorPos = iLength - 3;
                            break;
                default:    break;
            }
        }
    }while(cChar == 0);
    /* Fenster wieder schließen */
    CloseWindow(pcWinPtr);
    ShowFiles();
    return(cChar);
}
/* —— Daten löschen ———————————————— */
void    CleanUp()
{
    bModify = 0;
    iFileAnz = iHauptPrgPtr = -1;
    iFilePtr = 0;
    return;
}
/* —— Dateinamen mittels File-Requester abfragen ——— */
INT     FileReq(pcName, iMode)
CHAR    *pcName;            /* Dateiname */
INT     iMode;              /* Modus (0=Lesen, 1=Schreiben) */
{
    CHAR    *pcWinPtr,          /* Zeiger auf Fensterpuffer */
            acMode[10],         /* Modus-String */
            acInput[21],        /* Dateiname */
            acPath[21];         /* Pfad zur Datei */
    INT     iEscape;            /* Abbruch-Flag */

    if(iMode == 0)
        strcpy(acMode, "lesen");
    else
        strcpy(acMode, "schreiben");
    strcpy(acPath, "A:");
    strcpy(acInput, "DISVER.DIS");
    pcWinPtr = OpenWindow(3, 1, 35, 5);
    gotoxy(5, 3); printf("      DISVER-Daten %s", acMode);
    gotoxy(5, 4); printf("Pfad:      <%-20s>", acPath);
    gotoxy(5, 5); printf("Dateiname: <%-20s>", acInput);
    iEscape = Input(17, 4, 19, acPath);
    if (iEscape == 1){
        CloseWindow(pcWinPtr); return(iEscape);
    }
    iEscape = Input(17, 5, 19, acInput);
    CloseWindow(pcWinPtr);
```

```
    if (iEscape == 1) return(iEscape);
    ShowFiles();
    /* Dateinamen zusammenbasteln */
    strcpy(pcName, acPath); strcat(pcName, "\\");
    strcat(pcName, acInput);
    return(0);
}
/* —— Disver-Daten laden ——————————————— */
void    FileLaden()
{
    INT     iEscape,          /* Abbruch-Flag */
            iAnzahl,          /* Anzahl gelesener Bytes */
            iInput;           /* Filedeskriptor der Datei */
    CHAR    acInput[42],      /* Dateiname */
            acIdentify[7];    /* Kennung der Daten */

    iInput = 1;
    do{
        /* Dateinamen abfragen */
        iEscape = FileReq(acInput, MO_LESEN);
        if (iEscape == 1) return;
        /* Datei öffnen */
        iInput = open(acInput, O_BINARY);
        if(iInput == -1) Error(4);
    } while(iInput == -1);
    /* Kennung prüfen, ob es sich auch um DISVER-Daten handelt */
    iAnzahl = read(iInput, acIdentify, 7);
    if(iAnzahl < 7) return;
    if(strcmp(acIdentify, "DISVER") == 0){
        Status(203);
        /* Änderungs-Flag setzen, wenn Daten angehängt werden */
        if (iFileAnz != -1) bModify = 1;
        /* Daten einlesen */
        while(iAnzahl){
            iFileAnz++;
            if(iFileAnz > MAXFILES){
                iFileAnz-; Error(5);
                break;
            }
            /* Speicherplatz für den File-Eintrag reservieren */
            if (AllocEntry(iFileAnz, FILES) == 1){
                iFileAnz-; break;
            }
            iAnzahl = read(iInput, File[iFileAnz],
                sizeof(struct FileEntries));
        }
    }
    else{
        Error(6);
        return;
    }
    close(iInput);
    Status(0); iFileAnz-;
    iFilePtr = 0; ShowFiles();
```

```
    return;
}
/* —— DISVER-Daten speichern ——————————— */
void    FileSpeichern()
{
    CHAR    acOutput[42];     /* Dateiname */
    INT     i,                /* Zähler */
            iEscape,          /* Abbruch-Flag */
            iAnzahl,          /* Anzahl geschriebener Bytes */
            iOutput;          /* Filedeskriptor der Ausgabedatei */

    iOutput = 1;
    do{
        /* Dateinamen abfragen */
        iEscape = FileReq(acOutput, MO_SCHREIBEN);
        if (iEscape == 1) return;
        /* Datei öffnen */
        iOutput = _creat(acOutput, FA_ARCH);
        if(iOutput == -1) Error(7);
    } while(iOutput == -1);
    /* Kennung schreiben */
    iAnzahl = write(iOutput, "DISVER", 7);
    if (!iAnzahl){
        Error(8); return;
    }
    Status(204);
    for(i = 0; i <= iFileAnz; i++){
        iAnzahl = write(iOutput,File[i],sizeof(struct FileEntries));
        if(!iAnzahl){
            Status(0); Error(8);
            return;
        }
    }
    close(iOutput); Status(0);
    /* Änderungsflag zurücksetzen */
    bModify = 0; ShowFiles();
    return;
}
/* —— Programm beenden und vorher allen ——————
  —— belegten Speicher freigeben ———————— */
void    ThisIsTheEnd()
{
    INT     i;      /* Zähler */

    if (bModify == 1) FileSpeichern();
    for (i = 0; i <= iFileAnz; i++) free(File[i]);
    clrscr();
    printf("Vielen Dank für den Einsatz \nvon DISVER Portfolio.\n");
    exit(0);
}
/* —— Sortieren der File-Liste nach dem Filenamen ———
  —— im Bereich von 0 bis iEnd ——————— */
void    NameSort(iEnd)
INT     iEnd;           /* Ende des Sortierbereiches */
```

```
{
    INT     iUntereSchranke, iObereSchranke,
            i, j, k;                        /* Zähler */
    struct FileEntries *W, *X;              /* scratch pointer */

    iUntereSchranke = (iEnd / 2) + 1;
    iObereSchranke = iEnd;
    while (iObereSchranke > 0){
        if (iUntereSchranke > 0){
            iUntereSchranke--; i = iUntereSchranke;
        }
        else{
            W = File[0]; File[0] = File[iObereSchranke];
            File[iObereSchranke] = W;
            iObereSchranke--; i = 0;
        }
        X = File[i]; k = 0;
        while ((2 * i <= iObereSchranke) && (k == 0)){
            j = 2 * i;
                if (j < iObereSchranke){
                        if (strcmp(File[j]->acName,
                            File[j + 1]->acName) < 0)
                            j++;
                }
                if (strcmp(X->acName, File[j]->acName) < 0){
                    File[i] = File[j]; i = j;
                }
                else
                    k = 1;
        }
        File[i] = X;
    }
    return;
}
/* --- Sortieren der File-Liste nach dem Kartennamen -----
   --- im Bereich von 0 bis iEnd ----------------- */
void    CardSort(iEnd)
INT     iEnd;               /* Ende des Sortierbereiches */
{
    INT     iUntereSchranke, iObereSchranke,
            i, j, k;                        /* Zähler */
    struct FileEntries *W, *X;              /* scratch pointer */

    iUntereSchranke = (iEnd / 2) + 1;
    iObereSchranke = iEnd;
    while(iObereSchranke > 0){
        if(iUntereSchranke > 0){
            iUntereSchranke--; i = iUntereSchranke;
        }
        else{
            W = File[0]; File[0] = File[iObereSchranke];
            File[iObereSchranke] = W;
            iObereSchranke--; i = 0;
        }
```

```
        X = File[i]; k = 0;
        while((2 * i <= iObereSchranke) && (k == 0)){
            j = 2 * i;
            if(j < iObereSchranke){
                    if(strcmp(File[j]->acDisk,
                        File[j + 1]->acDisk) < 0)
                        j++;
            }
            if(strcmp(X->acDisk, File[j]->acDisk) < 0){
                File[i] = File[j]; i = j;
            }
            else
                k = 1;
        }
        File[i] = X;
    }
    return;
}
/* —— Sortieren der File-Liste nach der Größe ————
   —— in Bytes im Bereich von 0 bis iEnd ———————— */
void    ByteSort(iEnd)
INT     iEnd;           /* Ende des Sortierbereiches */
{
    INT     iUntereSchranke, iObereSchranke,
            i, j, k;                    /* Zähler */
    struct FileEntries *W, *X;          /* scratch pointer */

    iUntereSchranke = (iEnd / 2) + 1;
    iObereSchranke = iEnd;
    while(iObereSchranke > 0){
        if(iUntereSchranke > 0){
            iUntereSchranke—; i = iUntereSchranke;
        }
        else{
            W = File[0]; File[0] = File[iObereSchranke];
            File[iObereSchranke] = W;
            iObereSchranke—; i = 0;
        }
        X = File[i]; k = 0;
        while((2 * i <= iObereSchranke) && (k == 0)){
            j = 2 * i;
            if(j < iObereSchranke){
                    if(File[j]->lBytes < File[j+1]->lBytes) j++;
            }
            if(X->lBytes < File[j]->lBytes){
                File[i] = File[j]; i = j;
            }
            else
                k = 1;
        }
        File[i] = X;
    }
    return;
}
```

```
/* —— File editieren ——————————————————— */
void    EditAFile()
{
    CHAR    *pcWinPtr,              /* Zeiger auf den Fensterpuffer */
            acEditBytes[7];         /* Puffer für die Bytes */
    INT     i;                      /* Zähler */
    LONG    lEditBytes;             /* Puffer für die Bytes */

    /* Fenster öffnen und beschreiben */
    pcWinPtr = OpenWindow(3, 1, 34, 6);
    gotoxy(13, 3); printf("File editieren");
    gotoxy(6, 4); printf("   Name      Bytes    Karte");
    gotoxy(6, 5); printf("——————  ———  ——————");
    for(i = 0; i <= iFileAnz; i++){
        if(File[i]->cSelect == 1){
            /* Werte holen und anzeigen */
            gotoxy(6, 6);
            sprintf(acEditBytes, "%-ld", File[i]->lBytes);
            printf("%-12s %-6ld %-10s", File[i]->acName,
                File[i]->lBytes, File[i]->acDisk);
            /* Werte nacheinander abfragen */
            Input(6, 6, 12, File[i]->acName);
            do{
                Input(19, 6, 6, acEditBytes);
                lEditBytes = atol(acEditBytes);
                sprintf(acEditBytes, "%-ld", File[i]->lBytes);
            }while(lEditBytes == 0);
            File[i]->lBytes = lEditBytes;
            Input(26, 6, 10, File[i]->acDisk);
            /* Markierung löschen und Änderungs-Flag setzen */
            File[i]->cSelect = 0; bModify = 1;
        }
    }
    CloseWindow(pcWinPtr); ShowFiles();
    return;
}
/* —— Angeklickte Files löschen ——————————— */
void FileLoeschen()
{
    INT     i, j = -1;      /* Zähler */

    for(i = 0; i <= iFileAnz; i++){
        if(File[i]->cSelect == 0){
            /* Änderungen-Flag setzen */
            bModify = 1; j++;
            File[j] = File[i];
        }
        else
            File[i]->cSelect = 0;
    }
    iFileAnz = j; ShowFiles();
    return;
}
/* —— Neues File von Hand eingeben ——————————— */
```

```
void    FileNeu()
{
    INT     i;      /* Zähler */

    /* Prüfen, ob noch Platz für das neue File ist */
    if(iFileAnz < MAXFILES){
        /* Alle Markierungen löschen */
        for(i = 0; i <= iFileAnz; i++) File[i]->cSelect = 0;
        /* Leeres File anhängen */
        iFileAnz++;
        if (AllocEntry(iFileAnz, FILES) == 1){
            iFileAnz-; return;
        }
        /* Letztes File selektieren und editieren lassen */
        strcpy(File[iFileAnz]->acName, "");
        File[iFileAnz]->lBytes = 0;
        strcpy(File[iFileAnz]->acDisk, "");
        File[iFileAnz]->cSelect = 1;
        EditAFile();
        /* Ende der File-Liste zeigen */
        if(iFileAnz < 8)
            iFilePtr = 0;
        else
            iFilePtr = iFileAnz - 7;
        ShowFiles();
    }
    else
        Error(5);
    return;
}
/* -- Informationen über das Programm anzeigen ------- */
void    FileAbout()
{
    CHAR    *pcWinPtr;  /* Zeiger auf Fensterpuffer */

    /* Meldung ausgeben */
    pcWinPtr = OpenWindow(1, 0, 37, 7);
    if(pcWinPtr == NULL) return;
    gotoxy(10, 2); printf("DISVER Portfolio 1.0");
    gotoxy(10, 3); printf("von Michael Schuschk");
    gotoxy(7, 4);  printf("(C) 1990 Markt & Technik AG");
    gotoxy(3, 5);
    printf("%3d Files mit %d kB gespeichert.", iFileAnz + 1,
        do_sum() / 1024);
    gotoxy(5, 6);
    printf("Es ist noch Platz für %3d Files.", MAXFILES-iFileAnz-1);
    while(!kbhit()); CloseWindow(pcWinPtr);
    ShowFiles();
    return;
}
/* -- Markiertes File zum Hauptprogramm machen ------- */
void    HauptProgramm()
{
    INT     i;      /* Zähler */
```

```
    /* eventuelle vorherige Hauptprogrammmarkierung löschen */
    for (i = 0; i <= iFileAnz; i++)
        if (File[i]->cSelect == 2) File[i]->cSelect = 0;
    /* Markiertes File suchen */
    for (i = 0; (File[i]->cSelect == 0) && (i <= iFileAnz); i++);
    if (i != iFileAnz){
        /* File markieren und an HauptPrgPtr übergeben */
        iHauptPrgPtr = i; File[i]->cSelect = 2;
        lHauptPrgBytes = File[i]->lBytes;
        ShowFiles();
    }
    return;
}
/* —— Markierte Files zum Hauptprogramm addieren ——— */
void    UnterProgramm()
{
    INT     i, j;       /* Zähler */

    j = -1;
    /* Hauptprogramm markiert ? */
    if (iHauptPrgPtr == -1) return;
    for (i = 0; i <= iFileAnz; i++){
        if (File[i]->cSelect == 1){
            /* Unterprogramm addieren */
            lHauptPrgBytes += File[i]->lBytes;
            File[iHauptPrgPtr]->lBytes = lHauptPrgBytes;
            bModify = 1;
        }
        else{
            /* Hauptprogrammpointer anpassen */
            j++;
            if (i == iHauptPrgPtr) iHauptPrgPtr = j;
            strcpy(File[j]->acName, File[i]->acName);
            File[j]->lBytes = File[i]->lBytes;
            strcpy(File[j]->acDisk, File[i]->acDisk);
            File[j]->cSelect = 0;
        }
    }
    for (i = j + 1; i <= iFileAnz; i++) File[i]->cSelect = 0;
    iFileAnz = j;
    if (iFileAnz < 7)
        iFilePtr = 0;
    else
        iFilePtr = iHauptPrgPtr - 3;
    File[iHauptPrgPtr]->cSelect = 0;
    iHauptPrgPtr = -1; ShowFiles();
    return;
}
/* —— Aktuelle Systemzeit holen und umwandeln ——— */
CHAR    *Zeit()
{
    struct tm   *Tm;        /* von localtime() benutzt */
    LONG        lT;         /* von time() benutzt */
```

```
    static CHAR acZeitBuffer[20],  /* Systemzeit als String */
                acTage[7][3] = {"So","Mo","Di","Mi","Do","Fr","Sa"};

    time(&lT);
    Tm = localtime(&lT);
    sprintf(acZeitBuffer, "%s %2d.%2d.%4d %02d:%02d",
        acTage[Tm->tm_wday], Tm->tm_mday, Tm->tm_mon + 1,
        Tm->tm_year + 1900, Tm->tm_hour, Tm->tm_min);
    return(acZeitBuffer);
}
/* --- Daten zweispaltig drucken --------------- */
void    DruckenAlles()
{
    INT     i, j, k,         /* Zähler */
            iSeiten;         /* Anzahl zu druckender Seiten */
    FILE    *out;            /* Druckerhandle */

    /* Seitenzahl ermitteln */
    iSeiten = iFileAnz / 58 / 2;
    /* Kanal öffnen und Drucker auf 80 Zeichen/Zeile stellen */
    Status(200);
    out = fopen("lpt1", "w");
    if (out == NULL){
        Status(0); Error(9);
        return;
    }
    fprintf(out, "\x1bP");
    /* Files drucken */
    for(i = 0; i <= iSeiten; i++){
        fprintf(out,
         "         DISVER Magnetkartenverwaltung     %-s Seite %d von %d\n\n",
         Zeit(), i + 1, iSeiten + 1);
        fprintf(out,
         "             Name      Bytes   Karte                Name     Bytes   Karte\n");
        fprintf(out,
         "         ------ ---- -----          ------ ---- -----\n");
        for(j = 0; j <= 57; j++){
            k = (2 * i * 58) + j;
            if(k <= iFileAnz)
                fprintf(out, "         %-12s %6ld %-10s        ",
                    File[k]->acName, File[k]->lBytes,
                    File[k]->acDisk);
            else
                fprintf(out, "%-45s", " ");
            k = ((2 * i + 1) * 58) + j;
            if(k <= iFileAnz)
                fprintf(out, "%-12s %6ld %-10s\n", File[k]->acName,
                    File[k]->lBytes, File[k]->acDisk);
            else
                fprintf(out, "%-30s\n", " ");
        }
        /* Fußzeile und Seitenvorschub ausgeben */
        fprintf(out,
            "\n        Insgesamt sind %d Files mit %ld Bytes gespeichert\n%c",
```

```
            iFileAnz + 1, do_sum(), 12);
    }
    fclose(out); Status(0);
    return;
}
/* —— Projektmenü auswerten ———————————— */
void    ActionOnProjectMenu()
{
    CHAR    cKey;        /* Menüauswahl */

    cKey = Menu(ProjectMenu);
    switch(cKey){
        case 'n':   CleanUp(); ShowFiles(); break;
        case 'l':   FileLaden(); ShowFiles(); break;
        case 's':   FileSpeichern(); break;
        case 'k':   DisketteEinlesen(); SetupFileScreen();
                    iFilePtr = iFileAnz - 7;
                    ShowFiles(); break;
        case 'ü':   FileAbout(); break;
        case 'e':   bEnde = 1; break;
        default:    break;
    }
    return;
}
/* —— Sortiermenü auswerten ———————————— */
void    ActionOnSortMenu()
{
    CHAR    cKey;        /* Menüauswahl */

    cKey = Menu(SortMenu);
    switch(cKey){
        case 'n':   Status(205); NameSort(iFileAnz);
                    Status(0); break;
        case 'b':   Status(205); ByteSort(iFileAnz);
                    Status(0); break;
        case 'k':   Status(205); CardSort(iFileAnz);
                    Status(0); break;
        default:    break;
    }
    ShowFiles();
    return;
}
/* —— Editiermenü auswerten ———————————— */
void    ActionOnEditMenu()
{
    CHAR    cKey;        /* Menüauswahl */

    cKey = Menu(EditMenu);
    switch(cKey){
        case 'h':   HauptProgramm(); break;
        case 'u':   UnterProgramm(); break;
        case 'e':   EditAFile(); break;
        case 'n':   FileNeu(); break;
        case 'l':   FileLoeschen(); break;
```

```
        default:    break;
    }
    return;
}
/* —— Optionsmenü auswerten —————————— */
void    ActionOnOptionMenu()
{
    CHAR    cKey;       /* Menüauswahl */

    cKey = Menu(OptionsMenu);
    switch(cKey){
        case 'a':   strcpy(acDrive, "A:\\");
                    *OptionsMenu[2] = '√';
                    *OptionsMenu[3] = ' ';
                    *OptionsMenu[4] = ' ';
                    break;
        case 'b':   strcpy(acDrive, "B:\\");
                    *OptionsMenu[2] = ' ';
                    *OptionsMenu[3] = '√';
                    *OptionsMenu[4] = ' ';
                    break;
        case 'c':   strcpy(acDrive, "C:\\");
                    *OptionsMenu[2] = ' ';
                    *OptionsMenu[3] = ' ';
                    *OptionsMenu[4] = '√';
                    break;
        default:    break;
    }
    return;
}
/* —— Hauptmenü auswerten —————————— */
void    ActionOnMainMenu()
{
    CHAR    cKey;       /* Menüauswahl */

    cKey = Menu(MainMenu);
    switch(cKey){
        case 'p':   ActionOnProjectMenu(); break;
        case 's':   ActionOnSortMenu(); break;
        case 'e':   ActionOnEditMenu(); break;
        case 'd':   DruckenAlles(); break;
        case 'o':   ActionOnOptionMenu(); break;
        default:    break;
    }
    return;
}
/* —— Tastendruck auswerten —————————— */
void    ActionOnKey(cKey, cKeyExt)
CHAR    cKey,           /* ASCII-Code der Taste */
        cKeyExt;        /* erweiterter Tastaturcode */
{
    if(cKey != 0){
        switch(cKey){
            /* Leertaste - Markieren */
```

```
        case 32:    if (File[iFilePtr + iFileCursorPos]->cSelect
                        == 1)
                        File[iFilePtr + iFileCursorPos]->cSelect
                            = 0;
                    else
                        File[iFilePtr + iFileCursorPos]->cSelect
                            = 1;
                    ShowFiles();
                    break;
        /* H - Markiertes File zum Hauptprogramm machen */
        case 104:
        case 72:    HauptProgramm(); break;
        /* U - Markierte Files zum Hauptprogramm addieren */
        case 117:
        case 85:    UnterProgramm(); break;
        default:    break;
    }
  }
  else{
    switch(cKeyExt){
        /* F1 - Hauptmenü */
        case 59:    ActionOnMainMenu(); break;
        /* Cursor nach oben */
        case 72:    if(iFileCursorPos == 0)
                        FileScroll(UP);
                    else
                        iFileCursorPos-;
                    break;
        /* Cursor nach unten */
        case 80:    if(iFileCursorPos == 7)
                        FileScroll(DOWN);
                    else
                        if (iFilePtr+iFileCursorPos < iFileAnz)
                            iFileCursorPos++;
                    break;
        /* Bild nach oben (PageUp) */
        case 73:    FileScroll(PAGEUP); break;
        /* Bild nach unten (PageDn) */
        case 81:    FileScroll(PAGEDOWN); break;
        /* Pos1 (Home) */
        case 71:    FileScroll(HOME); break;
        /* Ende (End) */
        case 79:    FileScroll(END); break;
        default:    break;
    }
  }
  return;
}
/* —— Das Hauptprogramm ————————————
   —— (oder was erwarten Sie sonst am Ende des Quelltextes?) - */
INT     main()
{
  CHAR    cKey, cKeyExt;  /* ASCII- und erweiterter Tastencode */
```

```
#ifdef DEBUG
    /* Blockcursor einschalten */
    Regs.h.ah = 1; Regs.h.ch = 0;
    Regs.h.cl = 7;
    int86(0x10, &Regs, &Regs);
#endif
    /* File-Bildschirm aufbauen */
    SetupFileScreen();
    /* Variable initialisieren */
    iFileAnz = -1; iFilePtr = 0;
    bEnde = 0;
    /* Erkennung anzeigen */
    FileAbout();
    /* Taste holen und auswerten */
    do{
        GetKey(&cKey, &cKeyExt); ActionOnKey(cKey, cKeyExt);
        /* Anzahl der Files und freien Speicher anzeigen */
        gotoxy(35, 4); printf("%3d", iFileAnz + 1);
        gotoxy(34, 6); printf("%6u", coreleft());
        /* Cursor positionieren */
        gotoxy(2, iFileCursorPos + 1);
    }while(bEnde == 0);
    ThisIsTheEnd();
    exit(0);
}
```

3.3 C-Programmierung am Beispiel des Programms DISVER

Das komplette Programm zu erläutern, würde den Rahmen dieses Kapitels sprengen. Viele Teile sind auch Standardroutinen, die auf allen Rechnern ähnlich sind. Deswegen werden in den nächsten Abschnitten nur einige interessantere Teile exemplarisch besprochen. Die häufigen Kommentare innerhalb des Listings sollen Ihnen das Verständnis der nicht näher erläuterten Teile erleichtern.

3.3.1 Fenstertechnik

Was zeichnet die Bedienung aller Portfolio-Applikationen aus? Natürlich die Fenster- und Menütechnik. Auf die Menütechnik wird im nächsten Abschnitt eingegangen. Hier werden wir uns mit dem Grundlegenden beschäftigen. Denn: kein Menü ohne ein Fenster, jedenfalls auf dem Port-

folio. Allgemein muß ein Programm, das mit Fenstern arbeitet, folgende Funktionen bereitstellen:

- Bildschirmbereich sichern *(StoreWindow())*
- Bildschirmbereich löschen *(ClearArea())*
- Rahmen zeichnen *(DrawBorder())*
- gesicherten Bildschirmbereich zurückspeichern *(CloseWindow())*

Die entsprechenden Routinen von DISVER stehen in Klammern hinter den Punkten. Besprechen wir die einzelnen Funktionen in der obigen Reihenfolge. Im Listing finden Sie die Funktionen zu Beginn des Listings ab der Seite 278. Wenn Sie die Turbo-C-Funktionen durchstöbern, finden Sie auch die Funktion *window()*. Sie ist zusammen mit anderen Funktionen in der Header-Datei CONIO.H zusammengefaßt. Die Beschreibungen der einzelnen Funktionen lassen auf viel Erleichterung bei der Fensterprogrammierung hoffen. Leider zeigen die Befehle wenig Wirkung auf dem Portfolio. *cprintf()* zum Beispiel zeigt keine Ausgabe auf dem Bildschirm. Selbst wenn direkt nach dem Funktionsaufruf mittels DIP-Interrupt 61h der Bildschirminhalt refresht wird, passiert nichts auf dem Bildschirm. Hingegen funktionieren *clrscr()*, *getch()*, *gotoxy()* und *kbhit()* ohne Beanstandung. Funktionen, die ebenfalls nicht oder nur eingeschränkt funktionieren können, sind die *conio*-Funktionen, die Bildschirmattribute setzen.

StoreWindow() sichert den Inhalt eines rechteckigen Bereichs des Bildschirms in eine Puffervariable. Zuerst einmal müssen der Funktion die Koordinaten des Bildschirmbereichs übergeben werden. Das sind speziell die x- und y-Koordinate der linken oberen Ecke sowie die Breite und Höhe des Bereichs. Die Angabe der Breite und Höhe ist für die weitere Verarbeitung innerhalb der Funktion einfacher, als wenn der Bereich durch die linke obere und die rechte untere Ecke festgelegt würde. Aus den letzten Angaben wird die Größe des nötigen Pufferspeichers berechnet. Um auch die Koordinaten des Bereichs mitspeichern zu können, müssen zusätzlich vier Byte mehr reserviert werden. Anschließend werden die Koordinaten in den Pufferspeicher gebracht. Für die Sicherung des Bildschirminhalts ist es nötig, mit *MK_FP()* einen FAR-Zeiger in den Bildschirmspeicher zu definieren. Damit der Zeiger auf den richtigen Offset zeigt, muß auf die Organisation des Bildschirmspeichers eingegangen werden. Diese Organisation ist in allen Bildschirmmodi des Portfolio gleich und mit derjenigen der Monochrom-Karte des PC identisch. Der Speicher liegt im Segment B000h. Da jede Zeile 80 Zeichen enthält und für jedes Zeichen der ASCII-Code und das Attribut gespeichert werden, besteht jede

Zeile im Speicher aus 160 Byte. Deswegen berechnet sich der Offset eines Zeichens mit Hilfe der im Listing zu findenden Formel

Offset = (y * 160) + (x * 2)

Für jede Zeile des Bereichs wird der Offset der ersten Spalte berechnet, und ab dort werden doppelt soviele Bytes in den Pufferspeicher gebracht, wie durch die Breite des Bereichs angegeben sind. Abschließend muß nur noch der Zeiger auf den Pufferspeicher an die aufrufende Funktion zurückgegeben werden.

Das Löschen eines Bildschirmbereichs mit *ClearArea()* funktioniert ähnlich. Wenn der Zeiger in den Bildschirmspeicher analog zur vorigen Funktion berechnet ist, muß nur noch der entsprechende Wert direkt in den Speicher geschrieben werden. In diesem Fall soll der Bildschirm gelöscht werden, der Bereich wird also mit Leerzeichen gefüllt. Um das Ganze möglichst fix zu machen (der Portfolio ist ja nicht gerade der Schnellste), werden Zeichen und Attribut nicht einzeln geschrieben, sondern beide Werte zu einem Integer-Wert zusammengefaßt und geschrieben. Sie finden das Listing dieser Funktion auf Seite 279. Nun hat aber die Ansteuerung des LCD-Bildschirms eine Besonderheit. Wenn Sie direkt den Inhalt des Bildschirmspeichers ändern, sehen Sie danach keine Änderung in der Anzeige. Erst nach Aufruf der Funktion 12h des DIP-Interruptes 61h wird der Bildschirm refresht. Falls Sie die Programmentwicklung auf dem PC betreiben und das Programm auch dort ausprobieren und debuggen, bevor Sie es auf den Portfolio kopieren, stellt diese Besonderheit eine Gefahrenquelle ersten Ranges dar. Auf dem PC ist der Interrupt nicht belegt und verursacht beim Aufruf einen Systemabsturz, was nicht die feine englische Art ist. Leider werden beim Booten des PC die unbenutzten Interrupt-Vektoren auf 0000:0000 gelegt. Besser wäre, wenn sie auf einen IRET-Befehl zeigen würden. Dann würde bei Interrupt-Aufruf sofort wieder in das Programm zurückgekehrt, zwar ohne Wirkung, aber auch ohne Absturz. Um diese Hürde zu umgehen, ist zu Beginn des Listings das Symbol *DEBUG* definiert. Solange dieses Symbol definiert ist, wird durch bedingte Kompilierung der Interrupt-Aufruf übergangen. Erst wenn das Programm auf den Portfolio übertragen werden soll, wird das Symbol entfernt und damit der Interrupt aufgerufen.

Nachdem der Bildschirmbereich gesichert und gelöscht ist, fehlt zum Fenster nur noch der Rahmen. Er wird mittels *DrawBorder()* (im Listing ab Seite 279) gezeichnet. Auch hier wird wieder ein Zeiger auf die linke obere Ecke des Bildschirmbereichs gebraucht. Ab dieser Position werden dann nacheinander mit vier *for*-Schleifen die obere, rechte, untere und linke

Kante gezeichnet. Dazwischen dürfen natürlich die Ecken nicht vergessen werden. Um Programmzeilen zu sparen, sind innerhalb der Funktion direkt die ASCII-Werte der einzelnen Zeichen benutzt worden. Übersichtlicher wird es durch Verwendung von *#define*-Anweisungen, die die ASCII-Werte durch Symbole ersetzen, die besser verständlich sind. Am Ende der Funktion findet sich wieder das Refreshen des Bildschirms.

Damit Sie nicht immer alle drei Funktionen hintereinander aufrufen müssen, wird dies in der Funktion *OpenWindow()* zusammengefaßt. Konnte der Pufferspeicher reserviert werden, liefert die Funktion den Zeiger auf den Pufferspeicher zurück. Ist nicht genügend Speicher frei, wird der Bildschirminhalt nicht gesichert, kein Fenster geöffnet und ein NULL-Zeiger zurückgegeben.

Was geöffnet wurde, muß auch wieder zu schließen sein. Also ist eine Funktion *CloseWindow()* vonnöten, die ein offenes Fenster auch wieder schließt. Auf Seite 278 sehen Sie im Listing, daß das Schließen des Fensters nichts weiter ist als das Zurückschreiben des gesicherten Bildschirminhalts und abschließendem Bildschirm-Refresh.

Auf den Routinen *OpenWindow()* und *CloseWindow()* bauen die Funktionen *Error()* und *Status()* auf, die eine Fehler- bzw. Statusmeldung innerhalb eines neuen Fensters auf dem Bildschirm ausgeben.

Was hier im Programm realisiert wurde, können nur die grundlegendsten Routinen einer Fensterverwaltung sein. Sie können ein Fenster öffnen und wieder schließen. Doch was passiert, wenn Sie mehrere Fenster nacheinander öffnen und sie nicht wieder in der umgekehrten Reihenfolge schließen? Chaos auf dem Bildschirm natürlich. Was fehlt, sind Routinen, die genau dieses verhindern und die Fenster innerhalb eines Stapels verwalten oder auch ermöglichen, Fenster, die im Hintergrund liegen, in den Vordergrund zu bringen und umgekehrt. Solange Sie dieses nicht selbst programmieren, müssen Sie die Verwaltung der Fenster übernehmen und auf die obigen Probleme achten.

3.3.2 Menüprogrammierung

Nachdem wir jetzt die Fenster haben, soll im nächsten Schritt zur Menüprogrammierung übergegangen werden. Atari hat seinen eingebauten Applikationen eine weitgehend einheitliche Benutzerführung gegeben. Um die Gewöhnung an ein neues Programm möglichst einfach zu

machen, sollten alle selbsterstellten Programme ähnlich mit ihrem Benutzer kommunizieren.

Die Menüroutine sollte möglichst allgemein verwendbar sein und Fehlbedienungen von vornherein abfangen. Was braucht sie dazu an Informationen vom Programm? Zunächst natürlich, welche Punkte das Menü enthalten soll. Da die Menüauswahl alternativ auch über Eingabe von Buchstaben erfolgen soll, müssen auch diese bekannt sein. Damit hätten wir schon fast alle nötigen Informationen. Zusätzlich wird im Beispielprogramm noch der Menütitel mit hinzugenommen. Alle Informationen werden in einem zweidimensionalen CHAR-Feld zusammengefaßt und der Zeiger auf das Feld der Menüroutine als Argument mitgegeben. Sie finden gleich zu Beginn des Listings die Definitionen für alle Menüs des Beispielprogramms. Jedes Feld enthält zuerst einen String mit allen erlaubten Tastendrücken, dann den Titel des Menüs und zuletzt alle Menüpunkte. Nach Aufruf der Funktion *Menu()* wird das übergebene Menü angezeigt und der ASCII-Wert des Kennbuchstabens des gewählten Menüpunktes zurückgegeben. Anschließend muß nun die Menüauswertung folgen und Aktionen entsprechend dem Menüpunkt in die Wege leiten.

Was geschieht aber nun innerhalb der *Menu()*-Funktion? Schauen Sie sich dazu die Funktion im Listing ab Seite 297 an. Zuerst müssen die Koordinaten für das zu öffnende Fenster festgelegt werden. Die Höhe des Fensters wird durch die Anzahl der Menüoptionen festgelegt, die wiederum aus der Länge des Strings aller erlaubten Tasteneingaben bestimmt werden kann. Aus dem acht Zeilen hohen Bildschirm des Portfolio ergibt sich dann auch gleich die Maximalzahl von sechs Menüpunkten, da zwei Zeilen für den Fensterrand benötigt werden. Theoretisch wären auch längere Menüs möglich, aber dann befänden sich immer einer oder mehrere Menüpunkte außerhalb des Bildschirms und man müßte den Bildschirm scrollen, um alle Menüpunkte überblicken zu können. Die Übersichtlichkeit würde darunter leiden, und welcher Benutzer will schon gerne ständig hin- und herscrollen, um einen Punkt aus einem Menü auszuwählen.

Wenn man die Anzahl der Menüpunkte hat, ist es ein leichtes, den breitesten Menüpunkt zu suchen und die Breite des Fensters entsprechend größer zu wählen. Durch die zentrierte Ausgabe des Menüs ergeben sich automatisch die Berechnungsvorschriften für die Koordinaten der linken oberen Ecke des Fensters. Danach wird das Fenster geöffnet und der Menütitel zentriert in die obere Kante des Rahmens geschrieben. Die folgende *for*-Schleife gibt die Menüpunkte aus. Bevor die große Schleife mit

dem Holen und Auswerten von Tastendrücken beginnt, müssen noch zwei Variable auf ihre Anfangswerte gesetzt werden. *iMenuCursorPos* enthält, wie ihr Name schon sagt, die Nummer des Menüpunkts, auf dem der Cursor gerade steht. Sie wird zur Auswertung und richtigen Positionierung des Cursors gebraucht. *cChar* dient im weiteren zur Aufnahme des Rückgabewerts.

Innerhalb der Schleife wird der Cursor auf die erste Spalte des aktuellen Menüpunktes gesetzt, zu Beginn ist das der oberste. Als zweiter Schritt wird mit Hilfe der *GetKey()*-Funktion auf einen Tastendruck gewartet und der ASCII-Wert sowie der erweiterte Tastaturcode der gedrückten Taste geholt. Die Funktion findet sich im Listing auf Seite 280 und besteht im wesentlichen nur aus dem Aufruf der Funktion 0 des Tastatur-Interrupts 16h. Die Rückgabe des erweiterten Tastaturcodes ist notwendig, da die Betätigung der Cursor- oder anderer Sondertasten einen ASCII-Wert von 0 liefern und dadurch nicht zu differenzieren sind. Die Funktionen des Tastatur-Interrupt sind auf dem Portfolio identisch, so daß in diesem Punkt keine Schwierigkeiten bei der Umsetzung von PC-Programmen auftreten.

Anschließend wird ausgewertet, welche Taste gedrückt wurde. Relevant sind dabei nur die Tasten [Esc], [Return], [↑] und [↓]. Am einfachsten ist die Reaktion auf [Esc], es wird nur die Funktion mit der Rückgabe des Wertes 27 als ASCII-Wert der Taste beendet. Falls [↵] gedrückt wurde, muß bestimmt werden, auf welchem Menüpunkt der Cursor gerade steht und diejenige Taste in die Rückgabevariable gebracht werden, die im String der erlaubten Tasten mit dem gewählten Menüpunkt korrespondiert. Sollte eine andere Taste gedrückt worden sein, die einen ASCII-Wert ungleich 0 besitzt, wird mit der *strchr()*-Funktion gesucht, ob die Taste im String der erlaubten Tasten vorkommt. Ist dies der Fall, wird diese Taste in die Rückgabevariable kopiert und die Funktion beendet. Anderenfalls wird die Schleife erneut durchlaufen und eine neue Taste geholt. Wurde eine Taste gedrückt, deren ASCII-Wert gleich 0 ist, muß der erweiterte Tastaturcode ausgewertet werden. In diesem Zweig der Schleife wird nur die [↑]- oder [↓]-Taste geprüft. Entsprechend der Taste wird die aktuelle Cursorposition, die sich in der Variablen *iMenuCursorPos* findet, erniedrigt oder erhöht. Natürlich müssen die Positionen abgefangen werden, die den Cursor über den ersten oder unter den letzten Menüpunkt stellen würden. Verlassen werden kann die Schleife nur durch eine der oben vorgestellten Möglichkeiten. Als letztes wird das Fenster wieder geschlossen und der aktuelle Ausschnitt der Dateiliste neu angezeigt. Das erneute Anzeigen hat seinen Grund in möglichen Änderungen der Dateiliste durch Auswahl

eines Menüpunkts, die innerhalb des gesicherten Bildschirmbereichs noch nicht enthalten sind.

Eine mögliche Menüauswertungsroutine finden Sie innerhalb der Funktion *ActionOnProjectMenu()* (im Listing auf Seite 307), die das Hauptmenü des Dateibildschirms auswertet. Die *Menu()*-Funktion wird aufgerufen und der Rückgabewert *cKey* innerhalb der folgenden *switch*-Anweisung zum Verzweigen zu den verschiedenen Aktionen benutzt.

3.3.3 Tastatureingabe

Die Auswahl von Programmfunktionen mittels Menüauswahl ist nun möglich. Jetzt fehlt nur eine komfortable Eingabe von Zahlen- oder Zeichenketten über die Tastatur. C bietet zwar mit der *scanf()*-Funktion eine universelle Eingabefunktion, aber weder die Länge der Eingabe läßt sich begrenzen, noch ist ein weitergehendes Editieren der Eingabezeile möglich. Wenn Sie zum Beispiel eine Maske zur Eingabe einiger Werte in eine Datenbank programmieren wollen, zerstört jede Eingabe über *scanf()*, die länger als vorgesehen ist, Ihre Maske. Hier muß also eine eigene Routine diese Arbeit übernehmen.

Eine Möglichkeit, wie eine solche Routine aussehen könnte, liefert Ihnen die Funktion *Input()* ab Seite 280 im Listing. Die wichtigste Funktion, die für eine Tastatureingabe benötigt wird, ist das Auswerten eines Tastendruckes. In DISVER ist dies durch die Funktion *GetKey()* verwirklicht, die Sie im Listing direkt vor *Input()* finden. Die Funktion liefert dann den ASCII-Wert und den erweiterten Tastaturcode zurück. Entsprechend diesen beiden Werten muß dann reagiert werden. In der folgenden Tabelle sind die wichtigsten Tastaturcodes zusammengefaßt:

Taste	erweiterter Code
↑	72
↓	80
←	75
→	77
Bild↑	73
Bild↓	81
Pos1	71

Taste	erweiterter Code
Ende	79
Entf	83
Einfg	82
F1 – F10	59 – 68

Die Funktion *Input()* wird mit den Koordinaten der linken oberen Ecke des Eingabefeldes, der Breite des Feldes und einem String aufgerufen. Der String enthält bei der Rückkehr die eingegebene Zeichenkette und kann bei Aufruf der Funktion einen Vorgabewert enthalten, der nur noch editiert werden braucht. Der blinkende Blockcursor des Portfolio wird als Kennzeichnung dort positioniert, wo das nächste eingegebene Zeichen eingefügt wird. Damit kommen wir zu einem Merkmal der Eingaberoutine: Sie arbeitet nur im Einfüge-Modus, d.h. neu eingegebene Zeichen werden an der Cursorposition eingefügt und überschreiben die bereits vorhandenen Zeichen nicht. Eine mögliche Erweiterung der Routine könnte also darin bestehen, mit Hilfe der Einfg-Taste zwischen Einfüge- und Ersetzungs-Modus hin- und herzuschalten. Als Kennzeichnung des jeweiligen Modus könnte parallel zwischen einem Block- und einem Strichcursor umgeschaltet werden.

Gehen wir einmal die Eingaberoutine Schritt für Schritt durch. Zu Beginn wird Speicher für den Vorgabewert und die Eingabe reserviert. Da zu jedem C-String als Endemarkierung der ASCII-Wert 0 als Abschluß des String gehört und außerdem der Eingabecursor am Ende der Zeichenkette auch noch Platz braucht, müssen für beide Symbole zwei Byte mehr Platz belegt werden, als die Breite angibt. Nachdem der Vorgabewert in den gerade belegten Pufferspeicher gerettet wurde, wird ein »relativer Cursor« *iRelCursor* initialisiert, der die Position, relativ zum Beginn der Eingabezeile, enthält, an der das nächste Zeichen eingefügt wird. Der Cursor wird auf das erste Zeichen des Eingabefeldes gestellt und der Vorgabewert in den Eingabepuffer kopiert. Als letzte Vorbereitung überschreibt die Funktion das Eingabefeld mit Leerzeichen. Da zur Übersetzungszeit des Programms nicht bekannt ist, wie breit das jeweilige Eingabefeld ist, wird mit dieser Konstruktion der *printf()*-Argumente auch die Breite der Formatierung einer Variablen entnommen. Dies ist eine eher ungewöhnliche Konstruktion, die nicht so oft in C-Listings zu sehen ist, aber zeigt, was mit dem *printf()*-Funktion alles machbar ist.

Innerhalb der Eingabeschleife wird zuerst die Länge des String im Eingabepuffer bestimmt, um später bestimmen zu können, ob noch Platz für die weitere Eingabe eines Zeichens ist. Der aktuelle Inhalt des Eingabepuffers wird ausgegeben und der nächste Tastendruck geholt. Anschließend folgt die Auswertung der Tastencodes, getrennt nach Tasten, die einen ASCII-Wert zurückliefern und denjenigen, die nur einen erweiterten Tastaturcode besitzen. Auf die Auswertung wollen wir etwas näher eingehen. Am einfachsten läßt sich auf die Tasten [←], [→], [Pos1] und [Ende] reagieren. Der Eingabecursor wird so lange nach links bzw. rechts bewegt, bis er am Anfang bzw. Ende der Eingabezeile angelangt ist. Die Tasten [Pos1] und [Ende] bringen ihn direkt dahin.

Etwas mehr geistige Anstrengung muß bei der [Entf]-Taste aufgewandt werden. Was gemacht werden soll, ist sofort klar: Solange der Eingabecursor nicht hinter dem letzten Zeichen steht, wird das Zeichen unter dem Cursor gelöscht und alle rechts davon stehenden Zeichen werden herangezogen. In der Routine brauchen nur alle rechts vom Eingabecursor stehenden Zeichen um eine Stelle nach links kopiert werden. Turbo C stellt für diese Aktion zwei Funktionen zur Verfügung, *memcpy()* und *memmove()*. Beide kopieren Bytes von einer Ecke des Speichers zur anderen. Sollten sich aber Quell- und Zielbereich überlappen, muß man aufpassen. Liegt, wie hier, der Zielbereich vor dem Quellbereich, muß die Funktion *memcpy()* bemüht werden. Für den umgekehrten Fall, wenn der Zielbereich hinter dem Quellbereich liegt, ist dann *memmove()* zuständig. Damit ist die Auswertung der Tasten, die einen erweiterten Tastaturcode liefern, beendet.

Bei den Tasten, die einen ASCII-Wert besitzen, wird zwischen den Tasten [Esc], [Backspace], [Return], sowie allen anderen Tasten unterschieden. Das Drücken der [Esc]-Taste bewirkt das Kopieren des Vorgabewerts in den Eingabepuffer sowie das Setzen des Flags zum Verlassen der Schleife und des Flags zum Erkennen des Abbruchs. Wird die Eingabe durch Drücken der [Esc]-Taste abgebrochen, liefert *Input()* den Wert 1 zurück und enthält wieder den Vorgabewert im Eingabestring, der der Funktion am Anfang übergeben worden ist. Auf [Backspace] wird ähnlich reagiert, wie zuvor bei [Entf], bis auf den Unterschied, daß das Zeichen links vom Eingabecursor gelöscht wird. Die Argumente der *memcpy()*-Funktion sind deshalb auch fast die gleichen wie zuvor. Bei [Return] wird im Eingabepuffer das angehängte Leerzeichen wieder gelöscht und das Flag zum Verlassen der Schleife gesetzt. Im Gegensatz zu [Esc] bleibt hier der Rückgabewert in *iEscape* auf 0, um dem aufrufenden Programm mitzuteilen, daß die Eingabe ordnungsgemäß abgeschlossen wurde.

Bis jetzt wurden alle Sondertasten abgehandelt, aber es wurde nichts über die Tasten gesagt, die die Eingabe eigentlich ausmachen. Das geschieht nämlich erst hier am Ende der Schleife. Um die Eingabezeile nicht durch irgendwelche Steuerzeichen zu verunstalten, wird geprüft, ob der ASCII-Wert der Taste größer als 31 ist und ob noch Platz für ein weiteres Zeichen ist. Damit das neue Zeichen eingefügt werden kann, muß der Teil des Eingabepuffers um ein Zeichen nach hinten kopiert werden, der ab der Position des Eingabecursors steht. Hier kommt nun zum Tragen, was vorhin über das Kopieren von Speicherinhalten gesagt wurde. Würde mit Hilfe von *memcpy()* kopiert, bestünde der kopierte Teil des Eingabepuffers aus lauter gleichen Zeichen. Für diese Art des Kopierens muß die *memmove()*-Funktion verwandt werden. Der Unterschied beider Funktionen liegt in der Frage, von welchem Byte aus in welche Richtung der Quellbereich kopiert wird. Falls Sie diese Routinen lieber selber programmieren wollen, finden Sie im anschließenden Listing einen Vorschlag.

```
void *memmove(void *dest, const void *source, size_t number)
{
    INT   i;              /* Zähler */
    CHAR  *src, *des;     /* Zeiger auf Quell- und Zielbereich */

    src = (CHAR *)source;
    des = (CHAR *)dest;
    for (i = 0; i < number; i++)
        *dest[i] = *source[i];
    return(*dest);
}

void *memcpy(void *dest, const void *source, size_t number)
{
    INT   i;              /* Zähler */
    CHAR  *src, *des;     /* Zeiger auf Quell- und Zielbereich */

    src = (CHAR *)source;
    des = (CHAR *)dest;
    for (i = number - 1; i >= 0; i-)
        *dest[i] = *source[i];
    return(*dest);
}
```

Zum Schluß wird das neu eingegebene Zeichen an die gerade freigewordene Stelle kopiert. Sie können, wenn Sie wollen, weitere Zeilen in die *switch()*-Anweisung hängen, um auf andere Tasten reagieren zu können. Denkbar ist zum Beispiel ein weiterer Übergabeparameter, der es ermöglicht, nur noch Ziffern einzugeben und bei Eingabe anderer Zeichen-

tasten einen Warnton auslöst. Sie müssen dazu nur die *if()*-Anweisung hinter der *default:*-Marke entsprechend ändern.

Innerhalb der *Input()*-Funktion wird abschließend der Eingabepuffer angezeigt und in den Rückgabestring kopiert. Was bei sauberer Programmierung nicht fehlen darf, ist die Freigabe aller dynamisch belegten Speicherbereiche. Womit wir auch gleich beim Thema des nächsten Abschnitts wären, einer etwas intelligenteren Speicherverwaltung.

3.3.4 Speicherverwaltung

Gerade auf einem Rechner wie dem Portfolio mit wenig Hauptspeicher muß der Speicherplatz so effektiv wie möglich verwendet werden. DISVER belegt seinen benötigten Speicher dynamisch zur Laufzeit und führt eine Liste mit Zeigern auf die belegten Speicherbereiche. Gebraucht wird der Speicher für die Dateiliste, die Directory-Liste und eine interne Liste aller Unterverzeichnisse. Für jeden Eintrag wird Speicher belegt und der Zeiger auf den Bereich in den Listen *File[]*, *Dir[]* und *DirStack[]* abgespeichert. Das Problem besteht jetzt darin, daß C bei der Suche nach freiem Speicher nicht am Speicheranfang beginnt, sondern von der Position weitersucht, an der der letzte Bereich belegt wurde. Diese Methode der Speicherbelegung ist auch unter dem Name »Next fit« bekannt. An sich ist daran nichts Schlechtes, denn die Suche beginnt wieder am Anfang, wenn bis zum Ende kein passender Speicherblock mehr frei ist. Jedoch liefert die Funktion *coreleft()* falsche Werte über den verbleibenden Speicherplatz zurück. Anhand des restlichen Platzes können Sie aber sehen, ob noch Dateien eingelesen werden können. Also muß der Speicherplatz selbst so verwaltet werden, daß keine Lücken entstehen.

Nach dem Starten des Programms wird gewöhnlich die Dateiliste geladen und dabei für sie Speicher reserviert. Danach können die Directories eingelesen werden. Die dabei entstehende Directory-Liste benötigt natürlich auch Speicher. Zusätzlich wird für die Namen der einzelnen Unterverzeichnisse eine interne Liste angelegt, die genauso irgendwo abgelegt werden muß. Sobald aus dem Menüpunkt *Karte einlesen* zurückgekehrt wird, gibt DISVER auch den Speicher dieser beiden letzten Listen wieder für weitere Benutzung frei. Wird aber aus der Directory-Liste mindestens ein neuer Eintrag in die Dateiliste übernommen, wird Speicher benötigt, der hinter den beiden Listen reserviert wird. Kehrt man nun zum Dateibildschirm zurück, gibt DISVER den Speicher der Directory- und der

internen Liste zurück. Dieser Speicherblock ist wieder frei, kann aber nicht gleich wieder verwendet werden, da C, wie oben gesagt, ab der letzten Position weitersucht, an der Speicher belegt wurde und das ist, in diesem Beispiel, der neue Eintrag der Dateiliste. So schnell ist eine Lücke in der Speicherbelegung entstanden.

DISVER löst dieses Problem durch die Funktion *AllocEntry()*, die im Listing ab Seite 283 steht. Sie besteht aus drei nahezu identischen Teilen für die drei Listen *File[]*, *Dir[]* und *DirStack[]*, die die Zeiger auf die oben beschriebenen Speicherbereiche enthalten. Die Philosophie, die hinter der Routine steht, ist folgende. Am Anfang wird Speicher ganz normal mit *calloc()* belegt. Der Trick beginnt, wenn Speicher wieder freigegeben werden soll. Der Speicherbereich wird nämlich nicht an DOS zurückgegeben, sondern verbleibt belegt und wird erst am Ende des Programms komplett zurückgegeben. Fordert also DISVER Speicher für einen Listeneintrag an, wird *AllocEntry()* mit der gewünschten Position des Eintrags innerhalb der jeweiligen Liste und einer Kennziffer zur Unterscheidung der Listen aufgerufen. Danach wird geprüft, ob bereits vorher schon einmal Speicher für diesen Eintrag belegt worden ist und, ob die gewünschte Position nicht hinter dem Listenende liegt, das durch die Symbole *MAXFILES*, *MAXDIRS* bzw. *MAXDIRSTACKS* markiert ist. Treffen diese Bedingungen nicht zu, wird Speicherplatz für einen neuen Eintrag belegt. Ist innerhalb der Zeigerliste kein Platz mehr frei, erscheint eine Fehlermeldung.

Das vernünftige Arbeiten dieser Methode setzt aber voraus, daß innerhalb der Listen keine Lücken entstehen, so daß neue Einträge immer an das Ende der Liste gehängt werden müssen. *DISVER* sichert dies dadurch, daß der gelöschte Eintrag nicht einfach als gelöscht gekennzeichnet wird und trotzdem in der Liste verbleibt, sondern daß alle Einträge der Liste hinter dem zu löschenden Eintrag um eine Position nach vorne kopiert werden. In diesem Zusammenhang muß zwischen den Variablen *iFileAnz* und *iFilesTop*, *iDirAnz* und *iDirTop*, sowie *iDirStackAnz* und *iDirStackTop* unterschieden werden. Die Variablen, die auf »Anz« enden, enthalten die Anzahl der Einträge, die sich momentan in der Liste befinden, während die Variablen, die auf »Top« enden, kennzeichnen, für wie viele Einträge bereits Speicher reserviert wurde. Dabei ist die letzte Zahl immer größer oder gleich der Anzahl der Listeneinträge.

Mit dieser Methode der Speicherverwaltung zeigt Ihnen *coreleft()* die richtige Anzahl freier Bytes im Speicher und Sie können selbst abschätzen, ob Sie noch Platz für weitere Listeneinträge haben.

3.3.5 Einbindung von Assembler-Routinen

Prinzipiell ist die Einbindung von Assembler-Routinen in C ähnlich einfach wie in Pascal. Es gibt dabei aber einige Stolpersteine, die zu beachten sind. Da seien als Beispiele zuerst einmal das Speichermodell und die Unterscheidung von Groß- und Kleinschreibung genannt.

Am Beispiel der im Pascal-Kapitel vorgestellten Routine zum Sichern und Zurückschreiben von Bildschirminhalten sollen die Unterschiede demonstriert werden. Bezüglich der Schreibweise ist es notwendig, allen Assembler-Funktionen einen Unterstrich (_) voranzustellen. Außerdem müssen Sie auf die genaue Schreibweise achten, da C im Gegensatz zu Pascal »test« und »TEST« als zwei vollkommen unterschiedliche Symbole wertet.

```
IDEAL          ; Ideal-Modus des Turbo Assemblers einschalten
MODEL SMALL    ; Speichermodell = SMALL wählen

; SAVE.ASM :  Speichert Bildschirmausschnitt in Pointer-Variable
;
; Autor:                Frank Riemenschneider
; Anpassung an Turbo C: Michael Schuschk

CODESEG

DEBUG equ 1

PUBLIC _Save
PROC _Save FAR
ARG spseg: word, spofs: word, groesse: word, x1: word, y1: word,
    x2: word, y2: word

   push bp
   mov bp, sp
   push ax
   push bx
   push cx
   push si
   push di
   push ds
   push es
   mov ax,[spseg]
   mov ds,ax
   mov di,[spofs]       ;Speicheradresse Pointer laden
   mov ax,[di+2]        ;Segment Speicherbereich holen
   mov es,ax
   mov di,[di]          ;Offset Speicherbereich holen
   mov ax,[groesse]     ;Größe reservierter Bereich
   cld                  ;immer inkrementieren
   stosw                ;speichern
```

```
   mov ax,[y1]
   mov bx, 160
   mul bx            ;Startadresse Video-RAM berechnen:
   mov bx,[x1]
   shl bx,1          ;Y1*160+X1*2
   add ax,bx
   mov si,ax         ;= Offset
   mov ax, 0B000h
   mov ds,ax         ;Segmentadresse Videoram
   mov ax,[x1]
   stosw
   mov ax,[y1]       ;Koordinaten abspeichern
   stosw
   mov ax,[x2]
   sub ax,[x1]       ;(x2-x1+1) Bytes in x-Richtung
   inc ax
   mov bx,ax         ;merken und
   stosw             ;abspeichern
   mov ax,[y2]
   sub ax,[y1]
   inc ax            ;(y2-y1+1) Bytes in y-Richtung
   stosw             ;abspeichern
   mov cx,ax         ;Als Zähler setzen
Loop1:
   push cx
   push si
   mov cx,bx        ;Anzahl x-Bytes holen
   rep movsw        ;Speicherworte kopieren
   pop ax
   add ax,160       ;Neue Zeile anfangen
   mov si,ax
   pop cx
   loop Loop1       ;Nächste Zeile
   pop es
   pop ds
   pop di
   pop si
   pop cx
   pop bx
   pop ax
   mov sp, bp
   pop bp
   ret
ENDP

; LOAD.ASM :  Lädt Bildschirmdaten aus Pointer-Variablen ein
;
; Autor:                Frank Riemenschneider
; Anpassung an Turbo C: Michael Schuschk
```

```
PUBLIC _Load
PROC _Load FAR
ARG  speseg: word, speofs: word

   push bp
   mov bp, sp
   push bx
   push cx
   push si
   push di
   push ds
   push es
   mov ax,[speseg]
   mov ds,ax
   mov si,[speofs]       ;Speicheradresse Pointer laden
   mov ax,[si+2]         ;Segment  Speicherbereich holen
   mov si,[si]           ;Offset Speicherbereich holen
   mov ds,ax
   cld
   lodsw                 ;Größe reservierter Speicherbereich holen
   push ax               ;und merken
   lodsw                 ;x-Koordinate holen
   mov bx,ax
   lodsw                 ;y-Koordinate holen
   mov cx, 160
   mul cx                ;Startadresse Video-RAM berechnen:
   shl bx,1              ;Y1*160+X1*2
   add ax,bx
   mov di,ax             ;= Offset
   mov ax, 0B000h
   mov es,ax             ;Segmentadresse Videoram
   lodsw                 ;Anzahl x-Bytes pro Zeile holen
   mov bx,ax             ;Als Zähler setzen
   lodsw                 ;Anzahl y-Bytes pro Zeile holen
   mov cx,ax
Loop2:
   push cx
   push di
   mov cx,bx             ;Anzahl x-Bytes holen
   rep movsw             ;Speicherworte kopieren
   pop ax
   add ax,160            ;Neue Zeile anfangen
   mov di,ax
   pop cx
   loop Loop2            ;Nächste Zeile
IFNDEF DEBUG
   mov ah,18             ;Bildschirm refreshen
   int 97
ENDIF
   pop ax                ;Bereichsgröße zurückliefern
   pop es
   pop ds
   pop di
   pop si
```

```
    pop cx
    pop bx
    mov sp, bp
    pop bp
    ret
ENDP

END
```

Der Körper der beiden Funktionen ist vorher identisch. Geändert hat sich nur Anfang und Ende der Routinen. Da C mehrere Speichermodelle kennt, ist es notwendig, daß die Assembler-Funktion das gleiche Modell benutzt wie das C-Programm. Die vereinfachte Schreibweise der *PROC*-Anweisung läßt der Turbo Assembler nur für das Modell *TPASCAL* zu, so daß bei allen anderen Modellen auf die *ARG*-Direktive zurückgegriffen werden muß, um auf die Argumente zugreifen zu können. Auch dafür sind die Zeilen

```
push bp
mov bp, sp
```

zu Beginn der Prozedur und

```
mov sp, bp
pop bp
```

am Ende nötig. Der Turbo Assembler ersetzt zur Übersetzungszeit die indirekte Adressierung der Argumente durch eine Adressierung relativ zum Basiszeiger BP.

Um die Module zu einem Programm zu verbinden, assemblieren bzw. kompilieren Sie die Quelltexte getrennt mit dem Turbo Assembler bzw. der Kommandozeilenversion des Turbo-C-Compilers, TCC.EXE, und binden mit *TLINK* die Objektdateien zu einer EXE- oder COM-Datei. Falls Sie mit der integrierten Entwicklungsumgebung arbeiten, müssen Sie eine Projektdatei anlegen und benutzen, die den Namen Ihres C-Quelltextes und den Namen der Objektdatei mit dem Assembler-Modul, jeweils in einer Zeile, enthält. Um das getrennte Assemblieren Ihres Assembler-Quelltextes kommen Sie auch hier nicht herum. Zwingend nötig ist bei der Übersetzung die Angabe der Option */ml*, damit der Turbo Assembler die exakte Schreibweise Ihrer Symbole übernimmt und sie nicht alle groß-schreibt.

Anhang A: Elementare Datentypen und Zahlendarstellung

A.1 Elementare Datentypen in Turbo Pascal

Datentyp	Wertebereich	Anzahl Byte im Speicher
ShortInt	–128...127	1 Byte (mit Vorzeichen)
Integer	–32768...32767	2 Byte (mit Vorzeichen)
LongInt	–2147483648...2147483647	4 Byte (mit Vorzeichen)
Byte	0...255	1 Byte (ohne Vorzeichen)
Word	0...65535	2 Byte (ohne Vorzeichen)
Boolean	true/false	1 Byte
Char	'a'...'Z'	1 Byte (genau 1 Zeichen)
Real	$2.9*10^{-39}...1.7*10^{38}$	6 Byte (Genauigkeit 11–12 Stellen)
Single	$1.5*10^{-45}...3.4*10^{38}$	4 Byte (Genauigkeit 7–8 Stellen)
Double	$5.0*10^{-345}...1.7*10^{308}$	8 Byte (Genauigkeit 15–16 Stellen)
Extended	$1.9*10^{-4951}...1.1*10^{4932}$	10 Byte (Genauigkeit 19–20 Stellen)
Comp	$-9.2*10^{18}...9.2*10^{18}$	8 Byte (Genauigkeit 18–19 Stellen)
String	'aa'...	Länge + 1 Byte (max. 255)
Zeiger	Adresse	4 Byte

Die Datentypen *Single*, *Double*, *Extended* und *Comp* sind nur mit einem numerischen Koprozessor ({$N+}) oder nach Einbindung des Koprozessor Emulators ({$N+,E+}) verfügbar.

A.2 Zahlendarstellung

A.2.1 Darstellung von Integer-Zahlen im Rechner

Integer-Zahlen werden entweder als Zahlen im Zweierkomplement oder als BCD-codierte Zahlen dargestellt.

Einerkomplement-Darstellung:
Hierbei können positive und negative ganze Zahlen in einem Wertebereich,

der abhängig ist von der Anzahl der verwendeten Bits, dargestellt werden. Um im Einerkomplement aus einer positiven Zahl eine negative zu machen, müssen alle Bits umgedreht werden. Man erhält so eine Bitfolge, die als negative Zahl vom Rechner interpretiert wird. Der Nachteil besteht in der Doppeldeutigkeit der 0, es gibt sie mit positivem und negativem Vorzeichen. Der passende Assembler-Befehl zur Bildung des Einerkomplements lautet NOT.

Zweierkomplement-Darstellung:
Hierbei können positive und negative ganze Zahlen in einem Wertebereich, der abhängig ist von der Anzahl der verwendeten Bits, dargestellt werden. Um im Zweierkomplement aus einer positiven Zahl eine negative zu machen, müssen alle Bits umgedreht und beim letzten Bit eine 1 dazuaddiert werden. Man erhält so eine Bitfolge, die als negative Zahl vom Rechner interpretiert wird. Der passende Assembler-Befehl zur Bildung des Zweierkomplements lautet NEG.

BCD-Darstellung:
Bei dieser Darstellungsweise wird jede Ziffer einer Dezimalzahl als 4-Bit-Dualzahl codiert und im Rechner abgelegt.

A.2.2 Darstellung von Fließkommazahlen

Fließkommazahlen werden immer in der Form Vorzeichen/Exponent/ Mantisse im Rechner abgelegt. Je nach Genauigkeit und Wertebereich wird die Größe der Felder Exponent und Mantisse verändert. Die Zahl wird immer so umgeformt, daß die Basis für den Exponent eine 2 ist und die Mantisse mit 1,... beginnt (normalisierte Darstellung). Beim Ablegen der Mantisse wird die 1 vor dem Komma nicht mit abgespeichert. So wird ein Bit gespart. Das Feld Exponent bekommt einen Wert, der, abhängig vom Darstellungstyp der Zahl, durch Addition mit einer Konstanten gebildet wird (Single +127, Double +1023, Extended +16383). So sind auch negative Exponenten möglich.

A.2.3 Zusammenfassung

Die Frage, wie eine Bitfolge im Speicher interpretiert wird, hängt immer davon ab, was vom Programm oder vom Programmierer erwartet wird bzw. vorgesehen ist. So kann z.B. eine 64-Bit-Folge entweder als eine Long-

Integer- oder als eine Double-Precision-Zahl interpretiert werden. Je nach Interpretation wird natürlich ein anderer Zahlenwert weiterverarbeitet.

Eine Übersicht über die Zahlenformate zeigt die nächste Tabelle:

Datenformate	Bereich	Genauigkeit	Darstellung im Speicher
Word Integer	$\pm 10^{4}$	16 Bits	Zweier Komplement; 15 … 0
Short Integer	$\pm 10^{9}$	32 Bits	Zweier Komplement; 31 … 0
Long Integer	$\pm 10^{18}$	64 Bits	Zweier Komplement; 63 … 0
BCD	$\pm 10^{\pm 18}$	18 Digits	S | X | d17 ... d0; 79 72 0
Single Precision	$\pm 10^{\pm 38}$	24 Bits	S | BE | Mantisse; 31 23 0
Double Precision	$\pm 10^{\pm 308}$	53 Bits	S | BE | Mantisse; 63 52 0
Extended Precision	$\pm 10^{\pm 932}$	64 Bits	S | BE | Mantisse; 79 64 0

Bild A.1: Übersicht der Zahlenformate

Erklärung der verwendeten Abkürzungen:

S: *Sign*, Vorzeichenbit
BE: *Biased Exponent*, der bezogene Exponent einer Fließkommazahl.
X: Ohne Bedeutung
d??: Ein Digit (4 Bit), d.h. die ??. Stelle einer BCD Zahl.

Adresse: Man unterscheidet dabei zwischen einer physikalischen und einer logischen Adresse. Jede Stelle im Speicher des Rechners hat eine Adresse, unter der sie angesprochen werden kann. Diese Adresse wird als physikalische Adresse bezeichnet. Unter einer logischen Adresse versteht man eine Adresse, die z.B. nur innerhalb eines Segments als Relativadresse verwendet wird (Inhalt eines Adreßregisters). Um eine Speicherstelle ansprechen zu können, muß sie immer erst in eine physikalische Adresse umgewandelt werden.

Array: Ein Feld, das über einen Index adressiert werden kann und aus mehreren Elementen desselben Datentyps besteht.

Assembler: Ein Programm, das Mnemonics in Maschinensprache übersetzt.

Ausdruck: Ein Ausdruck wird aus mehreren Operatoren und Operanden gebildet und muß bestimmten Syntaxregeln folgen.

Bindung: Beim Binden erhält das aufrufende Programm die Adresse der aufzurufenden Prozedur. Diese Adreßbildung kann zum einen beim Kompilieren und Binden/Linken des Programms (frühe Bindung) oder erst zur Laufzeit des Programms (späte Bindung, eine der Stärken der OOP zusammen mit der Vererbung) passieren.

BIOS: Basic-Input-/Output-System. Im BIOS-ROM sind Routinen enthalten, um das Betriebssystem laden zu können. Mit Hilfe des BIOS kann man auf Peripheriegeräte zugreifen, ohne deren speziellen Aufbau zu kennen.

BIOS-Interrupts: Interrupts des Rechners, mit deren Hilfe BIOS-Funktionen aufgerufen werden können.

Breakpoint: Eine Stelle, an der das Programm angehalten wird. Ein Breakpoint kann auch durch Angabe einer bestimmten Bedingung ausgelöst werden. Wenn er bei jedem Befehl ausgelöst wird, spricht man von einem globalen Breakpoint.

Compiler: Ein Programm, das einen Quelltext einer Hochsprache (z.B. Turbo Pascal) in Maschinensprache übersetzt.

COM-Datei: Eine ausführbare Datei, die keine Verschiebeinformation für den Lader enthält.

CPU: Central Processing Unit. Eine CPU ist die Zentraleinheit/der Prozessor eines Rechners.

CPU-Register: Schnelle Speicherstelle (Register) auf dem CPU-Chip.

CS:IP: Beschreibt den aktuellen Wert des Befehlszeigers (*Instruction Pointer*). Dieser wird gebildet durch den Inhalt des CS-(*Code-Segment-*)Registers und des IP-(*Instruction-Pointer-*)Registers.

Default-Wert: Vorgabewert in einem Programm. Wird dieser Wert nicht geändert, so wird er übernommen.

Destruktor: Eine Methode, die mit dem neuen reservierten Wort *DESTRUCTOR* deklariert wird. Hierbei wird die Größe eines Objekts direkt vor dem Entfernen dieses Objekts vom Heap, also zur Laufzeit des Programms, ermittelt.

Dialogbox: Der Benutzer macht seine Eingaben in einem bestimmten Fenster, der sogenannten Dialogbox. Dabei werden die letzten Eingaben in einer Liste gespeichert, so daß sie ohne viel Aufwand wiederholt werden können.

Disassembler: Ein Programm, das Maschinenbefehle in lesbaren mnemonischen Code umwandelt.

DMA: Direct Memory Access. Hierbei werden die Daten an der CPU vorbei zum anfordernden Gerät übertragen.

DOS: Disk Operating System. Ein hardwareunabhängiger Teil des Betriebssystems, der z.B. für die Verwaltung der Peripheriegeräte zuständig ist.

DOS-Interrupts: Interrupt(s) des Rechners, mit deren Hilfe DOS-Funktionen aufgerufen werden können.

EXE-Datei: Eine ausführbare Datei, die Verschiebeinformation für den Lader enthält.

Hardware-Interrupt: Ein Interrupt, der durch die Rechner-Hardware generiert wird, z.B. durch einen Timer.

Instanz: Eine Variable eines Objekttyps, in den meisten Fällen sind Instanz und Objekt dasselbe.

Interrupt: Bei einem Interrupt wird das gerade bearbeitete Programm unterbrochen und zu einer sogenannten Interrupt-Service-Routine verzweigt. Ist die Routine beendet, wird die Abarbeitung des unterbrochenen Programms wieder aufgenommen.

Joker(zeichen): Ein Platzhalter für einen oder mehrere Buchstaben, z.B bei MS-DOS die Zeichen »*« und »?«.

Kapselung: In einem Objekt werden Code und Daten miteinander verbunden. Diese Verbindung bezeichnet man als Kapselung.

Konstruktor: Eine Methode, die mit dem neuen reservierten Wort *CONSTRUCTOR* deklariert wird. Hierbei wird ein Objekt initialisiert, das virtuelle Methoden enthält. Ohne vorherige Konstruktor-Deklaration ergibt der Aufruf einer virtuellen Methode einen Laufzeitfehler.

Laufzeitfehler: Bei der Ausführung eines Programms ist ein Fehler entstanden. Das Programm wird dann abgebrochen und eine entsprechende Fehlermeldung ausgegeben.

Linker: Ein Programm, das aus einem übersetzten Programm, meist eine Datei mit der Erweiterung ».OBJ« (Objektdatei), ein ausführbares Maschinenprogramm erzeugt, indem es z.B. für eine EXE-Datei die Tabelle der Verschiebeinformationen mit dem eigentlichen Programm zusammenbindet.

Makro: Eine Zusammenfassung von Befehlen, die durch eine Bezeichnung repräsentiert werden, z.B. in einem Assembler-Quelltext. Beim Übersetzen ersetzt dann der Assembler diese Bezeichnung jedesmal durch die Befehle.

Menü(leiste): Eine Menüleiste befindet sich bei Turbo Pascal und Turbo C in der obersten Zeile und kann durch Drücken von [F10] aktiviert werden. Bei der Auswahl einer Menübezeichnung durch Drücken der Cursortasten oder durch Drücken von [ALT] und dem Anfangsbuchstaben erscheinen Pull-down-Menüs, die herunterklappen. Aus diesen Menüs können dann weitere Unterpunkte ausgewählt werden.

Methode: Eine in einem Objekt definierte Prozedur oder Funktion. Dabei wird zwischen statischen und virtuellen Methoden unterschieden.

Statisch: Diese Methode wird bei der frühen Bindung implementiert. Die Adresse der Methode wird bei der Kompilierung des Programms ermittelt wie bei normalen Pascal-Prozeduren und -Funktionen.

Virtuell: Diese Methode wird bei der späten Bindung implementiert. Die Adresse der Implementierung wird zur Laufzeit des Programms ermittelt. Hierfür sind in Turbo Pascal 5.5 spezielle Hilfsmittel vorgesehen.

Mnemonic: Eine abgekürzte, symbolische Schreibweise für einen Maschinenbefehl.

Nachkomme: Ein Nachkomme ist ein Objekttyp, der wie in einem Stammbaum von einem anderen Objekttyp geerbt hat.

Objekt: Ein Objekt ist eine Variable eines Objekttyps.

Objektdatei: Diese Datei (meistens mit der Endung .OBJ) entsteht nach einem Übersetzungsprozeß. Sie ist nicht ausführbar und enthält Informationen für den Linker zum Erzeugen einer ausführbaren Datei.

Objekthierarchie: Durch Vererbung können mehrere Objekte zusammenhängen. Dieser Zusammenhang wird durch die Objekthierarchie beschrieben. Diese Hierarchie wird im Objekthierarchie-Fenster grafisch angezeigt.

Objekttyp: Ein Objekttyp ist eine spezielle Struktur, ähnlich einem Pascal-RECORD, der außer Datenfeldern auch Programmcode (Prozeduren und Funktionen) enthält.

OOP: Objektorientierte Programmierung. Ein neuartiges Konzept zur Lösung komplexer Programmieraufgaben.

Operand: Die Operanden werden durch einen Operator miteinander verknüpft.

Operator: Ein Operator kennzeichnet eine Aktion, die ausgeführt werden soll, z.B. wird bei einem »+«-Zeichen eine Addition der Operanden ausgeführt.

Polymorphie: Wird in einem Objekt eine Funktion definiert, dann implementiert jedes Objekt, das diese Funktion geerbt hat, die Funktion in der für dieses Objekt erforderlichen Art und Weise.

Programmzähler: Enthält die Adresse des nächsten auszuführenden Maschinenbefehls. Siehe auch CS:IP.

Quelltext: Der Text eines Programms, meist eine reine ASCII-Datei.

Run time error: Siehe Laufzeitfehler.

Segment (register): Die physikalische Speicher-Adresse wird aus dem Inhalt eines Segmentregisters und dem Inhalt eines Adreßregisters gebildet. Das Segmentregister legt dabei den Beginn des Segments im Speicher fest, das Adreßregister gibt dann die Lage innerhalb dieses Segments an. Ein Segment kann nicht größer als 64 Kbyte werden.

Stack: Ein Bereich für Daten mit einer LIFO-(Last in/first out-) Struktur. Auf diesem wird z.B. die Rücksprungadresse bei einem Unterprogrammaufruf abgelegt.

Symbol: Ein Symbol ist eine Bezeichnung für z.B. eine Variable oder eine Funktion. Diese Bezeichnungen werden in den sogenannten Symboltabellen abgelegt. Durch die Wahl einer bestimmten Option beim Übersetzen und Binden des Programms wird an eine ausführbare Datei die Symboltabelle angehängt, so daß z.B. der Turbo Debugger auf die Symbole zurückgreifen kann.

Trace: Abarbeitung eines Programms im Einzelschrittmodus.

Vererbung: Bei der Definition eines Objekts wird dieses in einen sogenannten Objektstammbaum eingeordnet. Jedes Objekt erbt dabei den Code und die Daten der Vorfahren.

Verschiebbarkeit: Ein ausführbares Programm kann in der Regel an eine beliebige Adresse im Speicher geladen werden. Der Lader benötigt Informationen, welche Teile des Programms modifiziert werden müssen, um das Programm an eine beliebige Stelle im Speicher verschieben zu können.

Vorfahr: Dieser Objekttyp wird an einen Nachfahren vererbt. Wird in einer Deklaration eines Objekttyps ein weiterer in Klammern angegeben, so ist dieser der direkte Vorfahr dieses Objekttyps.

Anhang C: Benutzung der Begleitdisketten

Übertragen der lauffähigen Programme auf den Portfolio

Auf den Begleitdisketten sind alle im Buch vorgestellten Programme bereits in kompilierter Form, d. h. als EXE-Dateien vorhanden. Sie können direkt vom PC mittels Kartenlaufwerk auf eine RAM-Karte kopiert werden. Die meisten Programme nutzen den Portfolio-Speicher voll aus, so daß vor Ihrem Start die interne RAM-Disk des Portfolio möglichst klein gemacht werden muß. Geben Sie dafür den DOS-Befehl

```
FDISK 8 [Return]
```

ein. Dieser Befehl minimiert die Größe der internen RAM-Disk auf 8 Kbyte. Anschließend kann das gewünschte Programm von der RAM-Karte gestartet werden, z. B.:

```
A:BEISPIEL [Return]
```

Neukompilierung der Programme

Wenn Sie irgendwelche Änderungen am Quelltext der Programme vorgenommen haben, müssen Sie diese natürlich auch neu kompilieren bzw. bei Assembler-Programmen neu assemblieren. Als Ziellaufwerk für die EXE-Datei sollte man das Kartenlaufwerk für die RAM-Karten angeben. Die neu entstandenen EXE-Dateien sollten dann mit dem Programm Portmaker bearbeitet werden, damit sie mit Sicherheit auf dem Portfolio zu starten sind. Anschließend können die Programme wie oben beschrieben gestartet werden.

Übertragen der lauffähigen Programme auf den Portfolio

Auf den Begleitdisketten sind alle im Buch vorgestellten Programme bereits in kompilierter Form, d. h. als EXE-Dateien vorhanden. Sie können diese vom PC mittels Kartenlaufwerk auf eine RAM-Karte kopiert werden. Die meisten Programme nutzen den Portfolio-Speicher voll aus, so daß vor ihrem Start die interne RAM-Disk des Portfolio möglichst klein gemacht werden muß. Geben Sie dafür den DOS-Befehl

FDISK 8 [Return]

ein. Dieser Befehl minimiert die Größe der internen RAM-Disk auf 8 Kbyte. Anschließend können die gewünschten Programme von der RAM-Karte gestartet werden, z. B.:

A:\BEISPIEL [Return]

Neukompilierung der Programme

Wenn Sie irgendwelche Änderungen am Quelltext der Programme vorgenommen haben, müssen Sie diese natürlich auch neu kompilieren bzw. bei Assembler-Programmen neu assemblieren. Als Ziellaufwerk für die EXE-Datei sollte man das Kartenlaufwerk für die RAM-Karten angeben. Die neu entstandenen EXE-Dateien sollten dann mit dem Programm Portfolio übertragen werden, damit sie mit Sicherheit auf dem Portfolio zu starten sind. Anschließend können die Programme wie oben beschrieben gestartet werden.

Anhang D: Hinweise auf weiterführende Literatur

Turbo Pascal Bediener-, Referenzhandbuch und Ergänzungsteil zur objektorientierten Programmierung, Borland International Inc.

Die Nachschlagewerke zu den vorhandenen Funktionen. Der Ergänzungsband liefert eine Einführung in die objektorientierte Programmierung.

Turbo Pascal 5.5 Schnellübersicht, Steiner, Markt & Technik Verlag, ISBN 3-89090-881-0

Wenn Ihnen die Originalhandbücher zu dick sind, können Sie hier schnell die gewünschten Informationen finden.

Effektives Programmieren mit Turbo Pascal, Kolbeck, Markt & Technik Verlag, ISBN 3-89090-771-7

Wenn Sie in Turbo Pascal maschinennah programmieren wollen, sollte dieses Buch das Richtige für Sie sein. Hier seien nur als Schlagworte TSR und DFÜ genannt.

Turbo Assembler Bediener- und Referenzhandbuch, Borland International Inc.

Die Nachschlagewerke zum Turbo Assembler. Im Bedienerhandbuch finden sich wertvolle Hinweise zur Assembler-Programmierung, speziell auch für den MASM-Modus.

Turbo Assembler Schnellübersicht, Soltendick-/Schuschk-/Riemenschneider, Markt & Technik Verlag, ISBN 3-89090-302-9

Die Schnellübersicht bietet eine zusammenfassende Sprachreferenz für den Turbo Assembler und Debugger. Für ein kurzes Nachschlagen kommen Sie mit einem statt mit drei Büchern aus.

PC-Programmierung in Maschinensprache, Monadjemi, Markt & Technik Verlag, ISBN 3-89090-503-X

Falls Ihnen die Einführung in die Assembler-Programmierung im Turbo-Assembler-Bedienungshandbuch nicht umfangreich genug war, können Sie in diesem Buch weitaus mehr dazu finden.

Turbo C Bediener- und Referenzhandbuch, Borland International Inc.

Die Nachschlagewerke zum Turbo C. Im Referenzhandbuch findet man die Auflistung aller C-Funktionen. Beide Werke sind meiner Ansicht nach besser zum Lernen von Turbo C geeignet als so manches Buch.

C in Beispielen, Huckert, Markt & Technik Verlag, ISBN 3-89090-466

Enthält eine umfangreiche Bibliothek an Routinen, die sofort in eigene Programme übernommen werden können.

Turbo-C 2.0 Schnellübersicht, Haselier, Markt & Technik Verlag, ISBN 3-89090-215

Wenn Ihnen die Originalhandbücher zu dick sind, können Sie hier schnell die gewünschten Informationen finden.

PC Intern, Tischer, Data Becker, ISBN 3-89011-331-1

Vom Umfang und Inhalt her, die »Bibel« für den PC-Programmierer.

Turbo Pascal Intern, Tischer, Data Becker, ISBN 3-89011-374-5

Ein Muß für jeden Turbo-Pascal-Programmierer.

Das 8086/8088 Buch, Rector/Alexy, te-wi Verlag, ISBN 3-921803-11-X

Das Referenzhandbuch für diesen Prozessor. Es enthält nicht nur die Beschreibung aller Befehle, sondern auch Timing-Diagramme und weitere Informationen über die Architektur dieses Prozessors.

Stichwortverzeichnis

Portfolio – Programmierpraxis

Zum Thema:

Der Atari Portfolio hat sich zu einem der verbreitetsten Pocket-Computer entwickelt. Insbesondere seine DOS-Kompatibilität und die mitgelieferten Standardprogramme zur Datenverwaltung, Textverarbeitung und Kalkulation machen diesen »Kleinstrechner« zu einem vielfältig einsetzbaren Werkzeug. Sein Funktionsumfang kann durch zusätzliche Software auf Speicherkarten erheblich erweitert werden. Die Software-Entwicklung für den Portfolio steckt heute jedoch noch weitgehend in den Kinderschuhen. Dies mag daran liegen, daß die Hardware sowohl Hobby-Programmierern als auch Entwicklern weitgehend unbekannt ist.

Buchhinweis:

Bei Markt&Technik erschienen:
Andreas Grote
Das Portfolio Praxisbuch
Einsatz – Beschreibung – Anwendung
1990, 231 Seiten
Bestellnummer: 90335

Aus dem Inhalt:

Ziel dieses Buchs ist es, das Rätselraten um das Innenleben des Portfolio zu beenden und zu zeigen, wie der Portfolio effektiv programmiert wird.

Im ersten Teil werden die technischen Details des Portfolio beschrieben. Vom Adreßraum, den Interrupts über den Bildschirm und die Tastatur bis hin zur seriellen Schnittstelle und zu den RAM-Speicherkarten werden die Unterschiede zu den »großen« PCs offengelegt.

Der zweite Teil zeigt, wie die »Programmier-Aufgabe Portfolio« mit Pascal und Assembler gelöst wird. Zur Vereinfachung der Bildschirmprogrammierung im Text- und im Grafik-Modus enthält das Buch Units, die Sie in Ihre Programme einbinden können. Außerdem werden ein Backup-Programm für Speicherkarten und ein Capture-Programm vorgestellt. Anhand des letzteren wird die Programmierung speicherresidenter Programme für den Portfolio demonstriert. Ein Schwerpunkt dieses Teils ist die Programmierung eines Chart- und Grafik-Programms, das auf dem kleinen LCD-Bildschirm die Darstellung von Balken-, Säulen und Kuchendiagrammen erlaubt.

Thema des dritten Teils ist schließlich die Programmierung mit C.

Am Beispiel eines umfangreichen Programms zur Verwaltung von RAM-Karten werden spezifische Programmiertechniken wie Tasten- und Menüprogrammierung erläutert.

Insgesamt ist die »Portfolio Programmierpraxis« ein ausführliches Lehr- und Nachschlagewerk zur Realisierung eigener Programmideen mit dem Atari Portfolio. Es beschreibt:

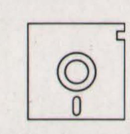
- Text- und Grafikprogrammierung
- TSR-Programme
- Backup für RAM-Speicherkarten
- Capture-Programm für LCD-Bildschirm

Systemanforderungen:

Hardware:
Atari Portfolio (Bios-Version 1.05 und höher).
PC/XT/AT oder Kompatible mit RAM-Kartenlaufwerk.
Mind. eine 64-Kbyte-RAM-Karte, mit der die Programme von den Disketten auf den Portfolio transferiert werden können.

Software:
Pascal, C und Assembler
(auf der Diskette zum Buch nicht enthalten)

Markt&Technik



ISBN 3-87791-020-3



DM 69,– sFr 64,– öS 538,–